D6.8 Crowd scene synthesis and metrics for quality evaluation

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	June 30th 2020
Actual delivery date	June 25th 2020
Leader	TCD
Reply to	Aljosa Smolic - smolica@scss.tcd.ie
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	**S**mart **A**sset re-**U**se in **C**reative **E**nvironments
Document name	D6.8 - Crowd scene synthesis and metrics for quality evaluation
Security (distribution level)	Public
Contractual date of delivery	June 30th 2020
Actual date of delivery	June 25th 2020
Deliverable name	Crowd scene synthesis and metrics for quality evaluation
Type	Demonstrator
Status & version	Submission version
Number of pages	49
WP / Task responsible	TCD
Other contributors	-
Author(s)	David Smyth, Amar Arslaan, Susheel Nath, Pisut Wisessing
EC Project Officer	Ms. Adelina Cornelia Dinu - adelina-cornelia.dinu@ec.europa.eu
Abstract	We provide a description of a prototype framework which can be used to rapidly generate a crowd simulation. The framework relies on the use of semantic data to feed into AI modules. We discuss how behaviour is separated from physical attributes (meshes and animation) which promotes re-use.
Keywords	Crowd simulation, AI, Semantic data
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	**Reason for Change**
1.0 03/06/2020	Document created.
1.1 15/06/2020	Version for internal review (14 days before submission date).
1.2 22/06/2020	Version for peer review.
1.3 25/06/2020	Updated with minor changes from peer review feedback.

Table of Contents

1 Executive Summary 5

2 Background 6

3 Introduction 7

3.1 Main Objectives and Goals, Relation to Self-Assessment Plan (D1.2) 7

3.2 Relationship to Other Deliverables 8

4 Implementation of the Crowd Simulation Prototyping System 8

4.1 Methodology 8

4.2 System Overview and Architecture 9

4.3 Scene Processing and Annotating 11

4.3.1 Semantic Data Annotation Using JSON-LD with Contexts 11

4.3.2 Semantic Animation Search User interface 13

4.4 Artificial Intelligence for Crowd Simulation 16

4.4.1 Potential Fields for High-Level Navigation 16

4.4.1.1 Specifying the Potential Function 17

4.4.1.2 Standalone Executable Interface Specification 19

4.4.1.3 Waypoint Length Reduction via Post-Processing 20

4.4.1.4 Interface With Semantic Data 21

4.4.1.5 Potential Field Goal Chaining 21

4.4.1.6 Waypoint Editing 22

4.4.1.7 Unity Implementation 23

4.4.2 Local Avoidance Implementation 24

4.4.3 Integration of Animation Triggers 25

4.4.4 Motion Stylization Implementation 26

4.4.5 Object Pooling for Runtime Creation and Destruction 28

4.5 Use Cases 28

4.5.1 Social Distancing in an Open Square 29

4.5.2 Pedestrian Scene 36

4.5.3 Commuter Scene 38

4.6 Metrics and Evaluation 41

4.6.1 Evaluation Metrics and Methods for Laboratory Testing 41

4.6.1.1 Software and Hardware Specifications 41

4.6.1.2 Stress Testing 41

4.6.2 Description of Methodology for Expert Review 44
4.6.3 Modifications Due to Covid-19 Pandemic 44
4.6.4 Steps Involved in Expert Review 45
4.6.5 Measurements 45
5 Conclusion 46
6 References 47

1 Executive Summary

This deliverable represents the total contribution to WP6T6: *Crowd Scene Synthesis*. It builds on the foundation laid in D6.3: *A working framework to handle relationship contexts between scene and people.* The overarching goal of this deliverable is to document a toolset and framework that leverages state-of-the-art techniques for the automated prototyping of re-usable crowd simulations. Our toolset has been developed in close collaboration with project partners. To aid development and evaluation, we devised a number of use cases which demonstrate the core technologies developed and demonstrate the versatility and potential of the toolset.

The approach taken was modular, where we break down the process of developing a general crowd simulation into various modules, with interfaces between modules defining the system. An overview of the approach is provided in Section 4.2. The modules separate behaviour from physical attributes, allowing for re-use of behaviours. These modules comprise our main contribution and were implemented in the Unity game engine, but we have documented them in anticipation of their use in other platforms. The toolset strongly promotes reuse of crowds through semantic data, which corresponds to a key theme of the SAUCE project. This has allowed us to collaborate with other project partners to leverage their developed technologies.

In relation to the ***Self-Assessment Plan*** **(D1.2),** this deliverable is a complete report on the work done for WP6T6. The complete design and architecture of the system are provided and the details of the technical implementation are given in Section 4. We report on metrics and the planned evaluation in Section 5. This deliverable builds on the work initially set out in D6.3 and ties together work outlined in D4.1 *Semantic Search Framework,* D6.6 *Motion Stylization Implementation* , D5.3 *Basic capability to enable asset transform* and will contribute to D8.4 *Report on Experimental Production Scenario Results* and D8.5 *Combined Evaluation Report.*

The evaluation of this toolset by independent users has been planned in Section 4.6 and the details will be reported in D8.4 *Report on Experimental Production Scenario Results* and D8.5 *Combined Evaluation Report*. Preliminary feedback from project partners who have been involved in the pilot study suggests that this work has strong potential for use in virtual production pipelines. Our own experience of this work suggests that this is a promising avenue for crowd simulation research. We anticipate that this work may be extended and lead to further attempts to bridge the gap between recent advances in AI and the current approach in the virtual production industry, which can often be heavily manual. Figure 1 - Figure 3 show some stills that display some results achieved when using our toolset.

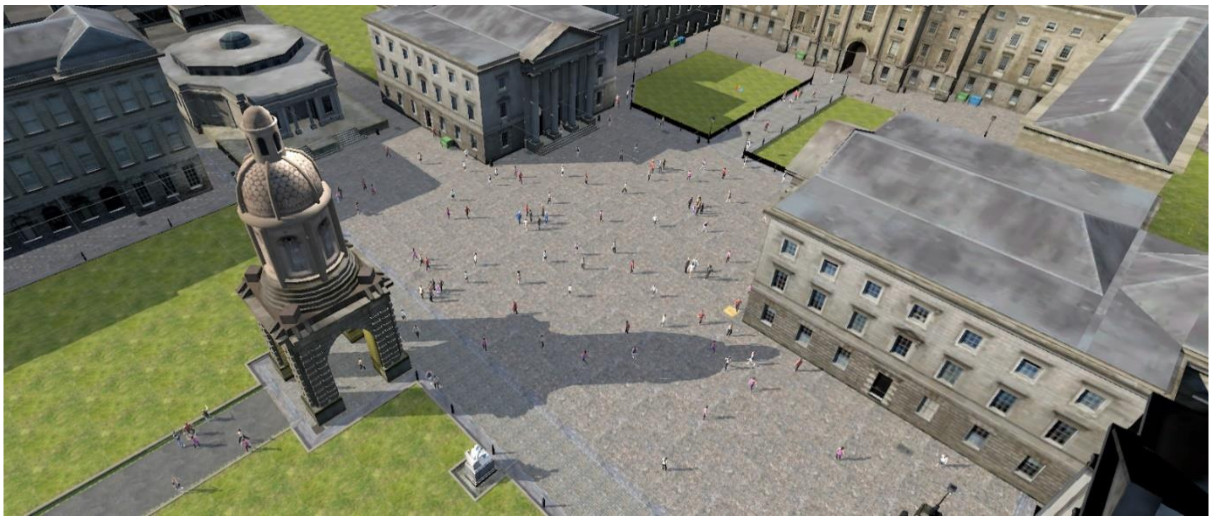


Figure 1 An example crowd scene developed using our toolset

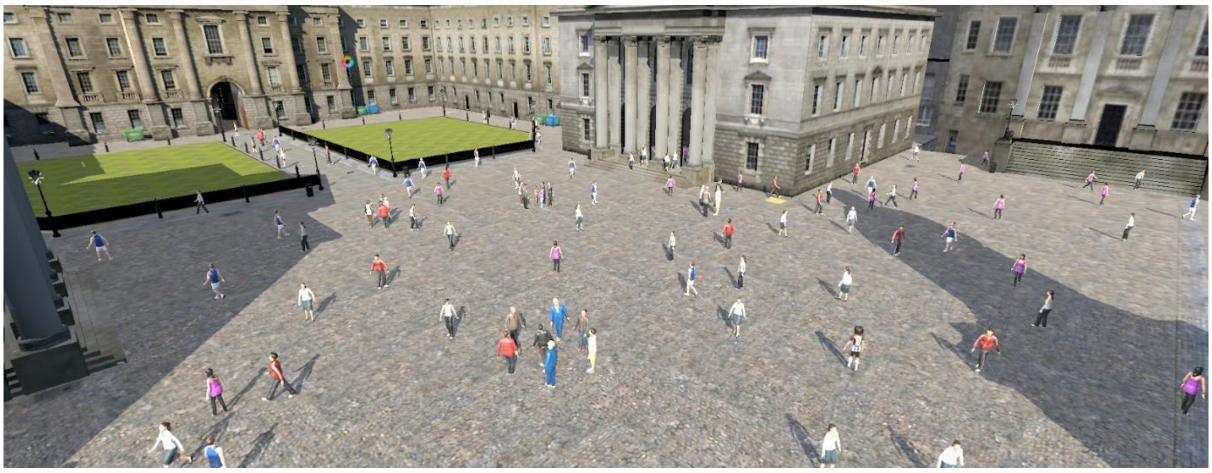


Figure 2

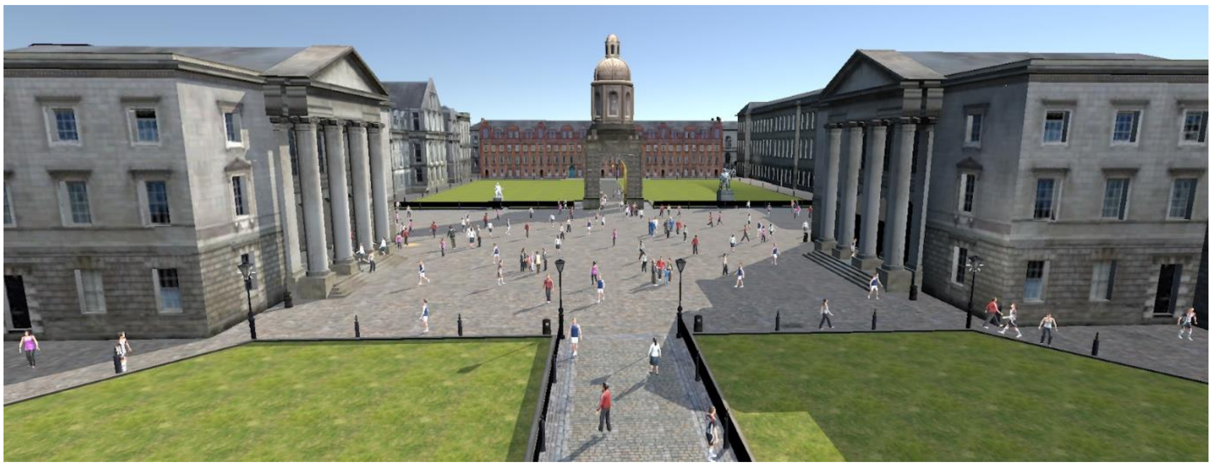


Figure 3

2 Background

The generation of crowd scenes for the film traditionally involves the hiring of extras, which can be costly from a monetary, time and organisational viewpoint. An in-depth discussion for the reasons behind this is provided in [1, p. 218] (we paraphrase from Section 12.1.1): The magnitude of the cost of crowd extras was aptly demonstrated in the film *Waterloo*, which employed over 15,000 extras but failed to even recoup its production costs. In more recent years, crowds in film scenes have turned virtual, with well-known examples including I am Legend [2], The Lord of the Rings [3] and World War Z [4]. A variety of standalone software and software suites have come into existence to help automate this task (Massive [5], Menge [6], UCrowds [7], etc.). These tools have also been used for general crowd simulations, which encompass evacuation scenarios, footfall modelling and pedestrian modelling for self-driving cars. In the last number of years, it has been identified that there is a demand for crowds that are reusable [8]: this means that the development of a crowd for one scene should not be a once-off process, but rather be something that can be adjusted to fit other similar scenarios, saving time and effort. This is explicitly described as a research aim in the SAUCE proposal abstract: "*Real-time control systems for authoring animated content using smart assets, automatically synthesizing new scenes from existing ones and integrating smart assets into virtual production scenarios* ". The main scope of this deliverable is to outline the toolset we have created, which can be used to develop crowds that are reusable through semantic data attached to their environments.

3 Introduction

This document outlines how the use of semantic data related to the environment can drive crowd behaviour, providing a way to de-couple crowd behaviour from their environment. This promotes a core concept of the SAUCE project: "*using smart assets which are adaptive to context, purpose, the user and the production environment*" [9]. The Background chapter has already introduced some of the key motivation behind the work. The remaining chapters are structured as follows:

The rest of this chapter sets out the main objectives and goals of the work and outlines the methodology behind our approach.

Chapter 4 (Implementation of the Crowd Simulation Prototyping System) provides the details of the implementation and the reasoning behind our approach. It is broken down into:

- System Overview and Architecture, which outlines the high-level design of the system, split into modular parts.
- Scene Processing and Annotating, which gives details of how scenes are processed and annotated with semantic data.
- Artificial Intelligence for Crowd Simulation, which gives the details of the technical implementation of the artificial intelligence modules developed to simulate crowd behaviour.
- Use Cases, which provides tangible examples of how the toolset can be effectively used for various scenarios. This builds on the use case initially proposed in D6.3 using LiDAR data. We outline three use cases:

    1. The first use case employs the use of assets developed in a previous crowd simulation project completed at TCD called Metropolis [10] and depicts a dense crowd scene in a large, open main square. Our toolset is used to demonstrate how "social distancing" can be implemented in this scene. We also leverage the semantic environment labels subsequently gathered for this environment, outlined in detail in D6.3.
    2. The second use case entails a scene in which groups make their way along a street, avoiding benches, streetlights and other pedestrians in a natural manner. The scene uses assets developed by Filmakademie as part of an experimental production entitled "Love and Fifty Megatons" [11].
    3. The final use case is the one initially described in D6.3. A group of commuters make their way down a street. This scene uses the LiDAR-constructed environment in D6.3. We re-target the crowd developed in the second use case to this scene, demonstrating the time-saving potential of our approach.

- Metrics and Evaluation, which contains the details of a planned user study, which will provide a way of objectively evaluating the effectiveness of the toolset. We also state which metrics we will collect and how we propose to collect them.

In the final Conclusions chapter, we highlight ways in which this work could be built upon and offer some conclusions on the important aspects of the system.

3.1 Main Objectives and Goals, Relation to Self-Assessment Plan (D1.2)

The primary objective of this deliverable is summarised in WP6T6: *"to employ semantic description of the assets and scenes to understand the relationship, context, and the distribution of people (crowd agents) in the scene. The process involves using this information to populate a new scene with crowd agents in believable situations and have them demonstrate expected behaviour."* This was motivated by a gap in existing technologies, identified by the SAUCE consortium.

The primary objective can be broken down into the following constituents, which we used to guide the progression of the work:

1. A systematic way to create or add semantic data at the scene level.
2. A systematic way to link semantic data to the crowd behaviour.
3. A systematic way to link crowd behaviour to meshes and animations.

The main goal of this work is to develop a system that offers a time-saving when developing a working prototype of a crowd simulation, achieved by re-targeting an existing crowd simulation. This is described in **D1.2:** ***Self-Assessment Plan***:

- Advance on the state of the art: ability to automatically populate new scenes based on examples taken from previously created and annotated scenes that are semantically similar, meeting metrics for quality evaluation defined in D6.8.
- Technology improvement: 50% time saving when editing the scene or quality of synthesis useful for pre-visualization or real-time application.

We comment on these goals and objectives in Section 4.6 and Section 5 in relation to our evaluation plan. We will conclude on whether we have achieved the goals and objectives once the evaluation of the system has been carried out and this will all be reported on in D8.4 *Report on Experimental Production Scenario Results,* and D8.5 *Combined Evaluation Report.*

3.2 Relationship to Other Deliverables

This deliverable makes use of the technologies developed in following SAUCE deliverables:

- D6.3: *A working framework to handle relationship contexts between scene and people*
- D4.1 *Semantic Search Framework*
- D6.6 *Motion Stylization Implementation*
- D5.3 *Basic capability to enable asset transform*

This deliverable will be evaluated in WP8 as part of:

- D8.4 *Report on Experimental Production Scenario Results*
- D8.5 *Combined Evaluation Report*

4 Implementation of the Crowd Simulation Prototyping System

4.1 Methodology

Our methodology broadly follows the reasoning presented in [12]. A crowd simulation should "expect to simulate the visual texture and contextual behaviours of groups of seemingly sentient beings" (P.1). A crowd can typically be described at a high-level by a global set of requirements (e.g. evacuation), but it is also noted that "computational methods for modelling one set of requirements may not mesh well with good approaches to another". (P.1). This intuitively means there is no "silver bullet" for computational crowd simulation: one approach will typically not work well for every realistic situation. This has led to several key general features being identified for a typical crowd, which reflect the areas in which research has addressed over recent years: *Appearance, Function, Time, Autonomy and Individuality.* [13]. These can be thought of as the primary axes along which crowd simulation research is carried out. This deliverable (and the SAUCE project as a whole) is focussed on the re-use of smart assets, therefore from this list we primarily address autonomy, time and function:

- Autonomy refers to how much manual intervention must be applied to control a crowd member's behaviour. A high level of autonomy implies little manual intervention. This can typically be achieved using artificial intelligence.
- Time specifically refers to how much time is needed to update each frame of the simulation. Real-time simulation is typically characterised as 24 fps or higher, in accordance with human perception.
- Function refers to how well a character can be calibrated to do certain tasks. This typically manifests itself in the rig configuration, cognitive models, joint limits, and other abilities such as animation-retargeting.

Intuitively, a re-usable real-time crowd system must have high levels of crowd member autonomy, must operate at > 24fps and must provide sufficient functionality per character to produce groups of "seemingly sentient beings". These three aspects allow us to address and achieve the sub-objectives in Section 3.1 in a structured manner.

In creative productions, *individuality* and *appearance* are addressed as a final step once the scene has been finalised, since they do not significantly affect the configuration of the scene. Once a crowd scene has been prototyped and finalised, higher-quality meshes and models are typically used for the fully-rendered production version. We therefore do not treat them as core aspects of this work, even though they appear on the list above.

Given this high-level approach, we provide details in Section 4 as to how we applied this methodology to the development process.

4.2 System Overview and Architecture

The crowd scene synthesis system developed to address the objectives for this deliverable was designed in a modular fashion to promote re-use and ease of creation of custom crowd simulations. This section serves as a guide in how these modular pieces work together, with subsequent sections providing the details of the technologies developed for each of the modules. As mentioned in the introduction section, we identified autonomy, time and function as key aspects in this work, which are reflected in the system architecture.

The high-level system architecture is shown in Figure 4. This figure is not exhaustive, but it shows the main components that have been developed for this deliverable. We emphasize the split between the scene and crowd members, which is a key factor for re-usability. The interfaces between modules developed for each of these are shown here. We factored out the semantic configuration of both the scene and people since configuration can be done as a pre-processing step.

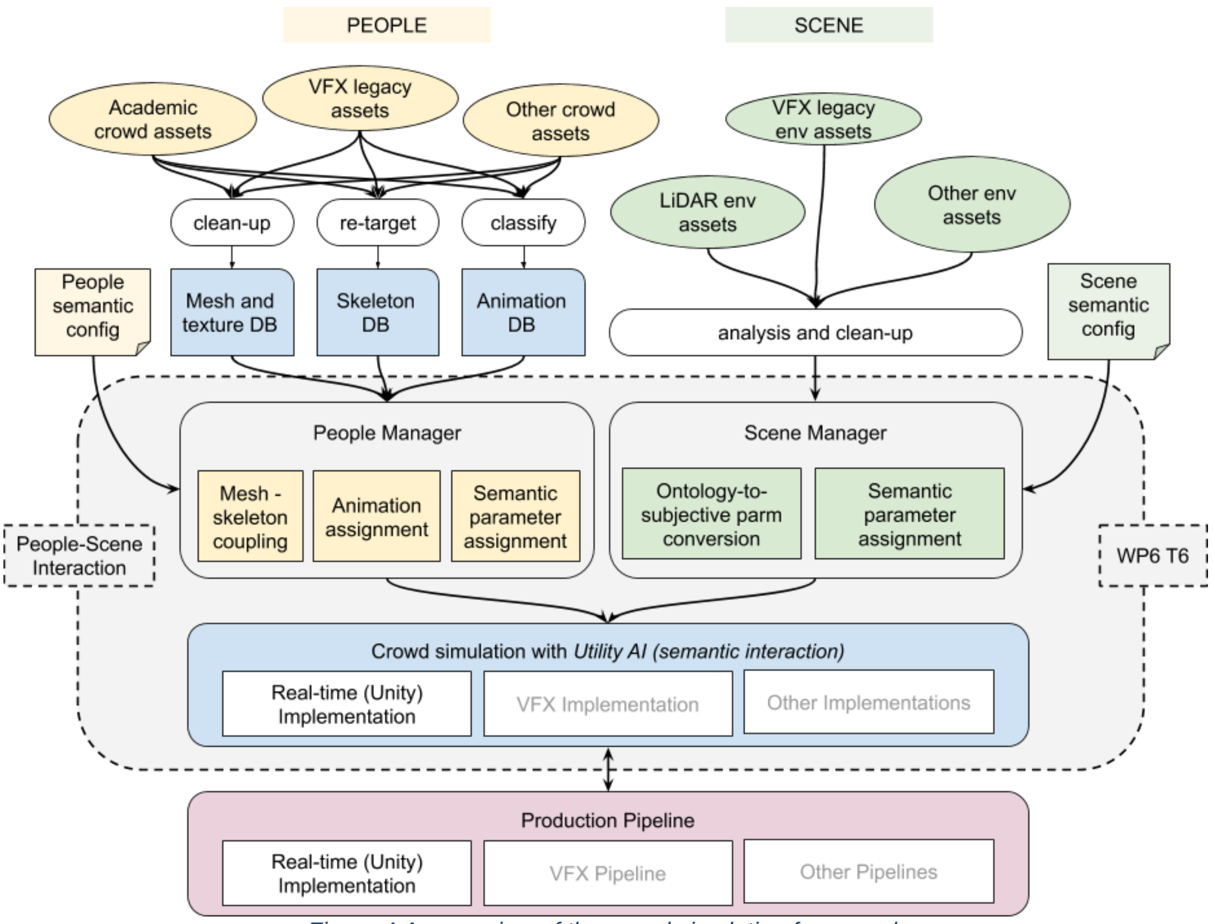
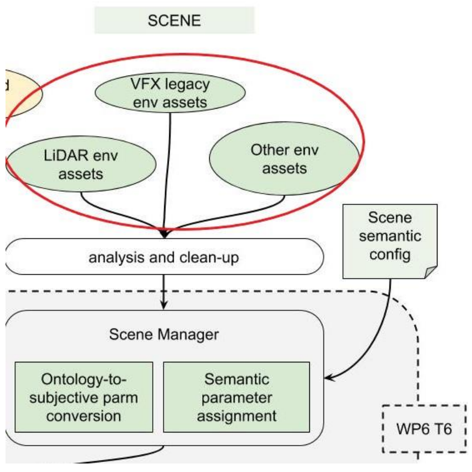
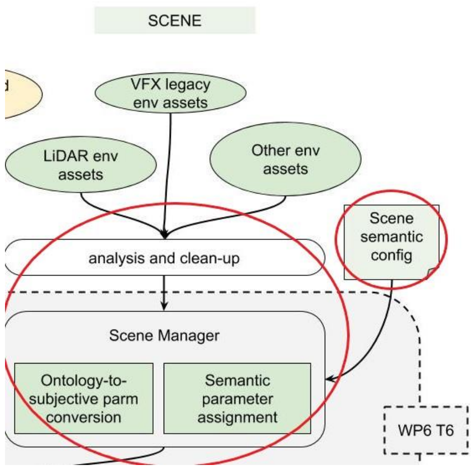


Figure 4 An overview of the crowd simulation framework

This diagram factorises a crowd scene into four main parts, which reflect the sub-objectives identified in Section 3.1. These correspond to the sub-figures shown in Figure 5.

1. The physical scene with accompanying semantic data.
2. The processing and configuration of the semantic scene data.
3. Crowd member physical parameters and animations.
4. The crowd behaviour.





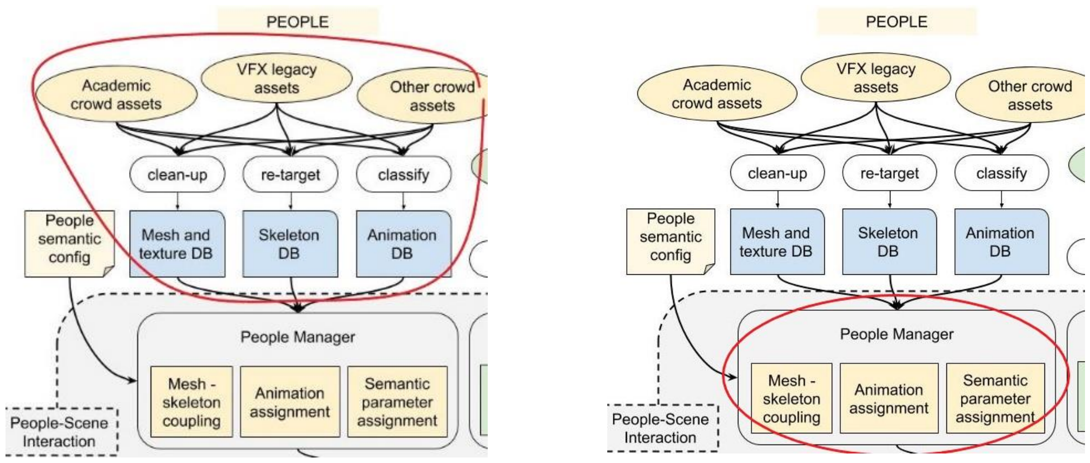


Figure 5 The main constituents of our crowd simulation framework

These parts feed into the real-time crowd simulation, which we have implemented using the Unity game engine [14].

4.3 Scene Processing and Annotating

Since our planned approach leverages the use of semantic data, we developed tools to help process existing scenes to annotate them with semantic data. We also developed several Unity-specific tools to which automate some tedious parts of running the simulation.

4.3.1 Semantic Data Annotation Using JSON-LD with Contexts

AI algorithms can be highly complex and difficult to interpret, therefore, to avoid nonsensical output that can be difficult to debug, the input parameters are required to be uniformly structured and well-defined. A key challenge presented by this work is how to transform the available semantic data into a uniformly structured and well-defined form that can be used as input to AI algorithms. This is illustrated in Figure 6, where the question mark represents the process of transforming arbitrary data sources to a common structure for AI modules.

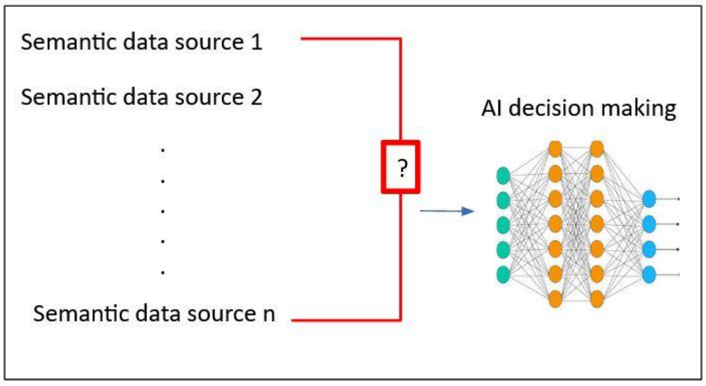
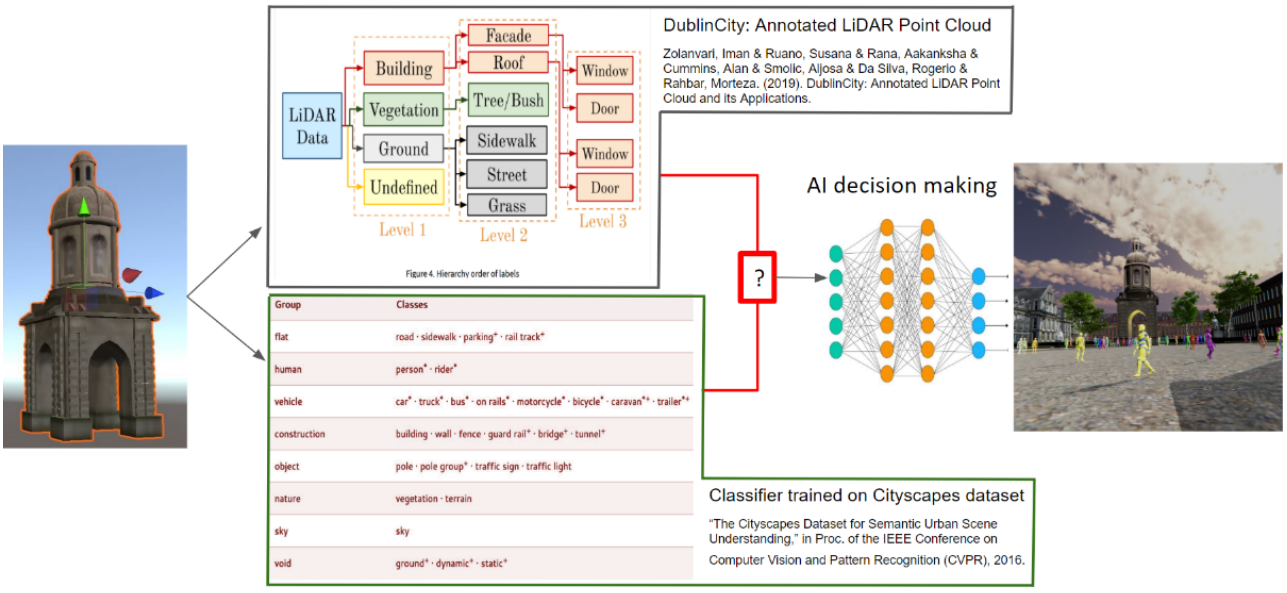


Figure 6 Semantic Data must be transformed to common structure for AI modules.

A tangible example of this is shown in Figure 7. Two different sources of semantically annotated data need to be transformed to a common structure to feed into an AI module. Although the labels are similar, the data have different hierarchies and labels.


DublinCity: Annotated LiDAR Point Cloud
Zolanvari, Iman & Ruano, Susana & Rana, Aakanksha & Cummins, Alan & Smolic, Aljosa & Da Silva, Rogerio & Rahbar, Morteza. (2019). DublinCity: Annotated LiDAR Point Cloud and its Applications.
LiDAR Data
Building
Vegetation
Ground
Undefined
Facade
Roof
Tree/Bush
Sidewalk
Street
Grass
Window
Door
Window
Door
Level 1
Level 2
Level 3
Figure 4. Hierarchy order of labels
?
AI decision making
Classifier trained on Cityscapes dataset
"The Cityscapes Dataset for Semantic Urban Scene Understanding," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

Group	Classes
flat	road · sidewalk · parking^{+} · rail track^{+}
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan^{*+} · trailer^{*+}
construction	building · wall · fence · guard rail^{+} · bridge^{+} · tunnel^{+}
object	pole · pole group^{+} · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground^{+} · dynamic^{+} · static^{+}

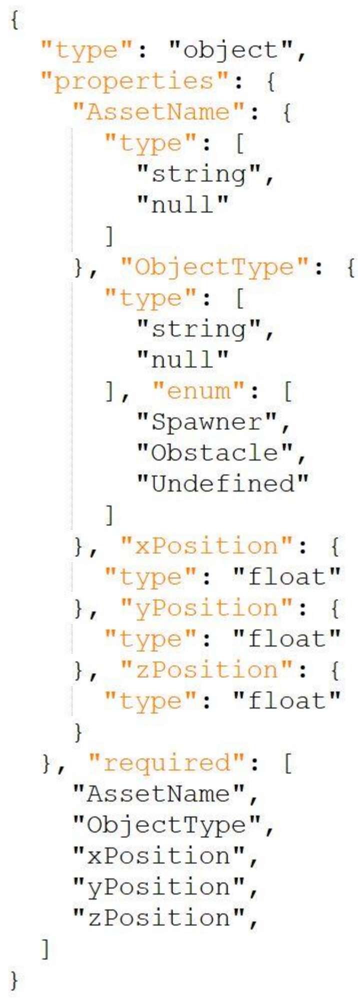
Figure 7 Semantic data must be structured in a common format when input to AI modules

We implemented this using JSON, with schemas and contexts. The main reasons for this are that JSON is readable and writable by both machines and humans, libraries exist in many languages to parse it and it can be serialized and distributed with minimal overhead. Consistency is also ensured using schemas, which can enforce constraints. The main drawback is that schemas must be manually defined.

To preserve the continuity and coherence of the work completed as part of SAUCE, we based our implementation on D4.1 Smart Search Framework, which in turn is based on D2.1 and D2.3. D4.1 documents the implementation of a mature and robust framework which supports the creation, extension and re-use of existing vocabularies and ontologies for arbitrary semantically annotated assets. We refer the reader to this D4.1 for an in-depth exposition. The Smart Search framework provides the ability to download assets along with related structured semantic data. We use the same structure, which is implemented using JSON with context provided by schemas.

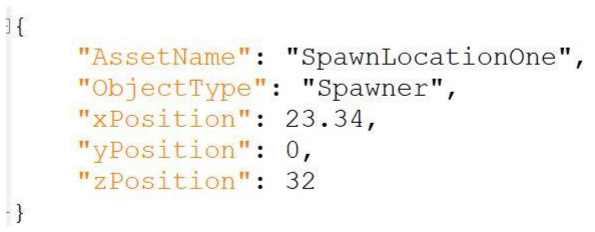
We developed a simple tool in the Unity editor which automates the creation of JSON files if they don't exist, for any asset in a given scene. The user can edit the parameters of the generated JSON files as desired directly from the Unity editor, or they can edit the file externally through a text editor or programmatically. The tool also allows JSON files to be validated against a schema inside the editor to ensure the structural integrity of the data. The data can be validated and then read by the AI algorithms to ensure that the input is valid. Section 4.4 outlines the main AI algorithms used, which require a well-defined input due to the high number of parameters.

A simple example is shown in Figure 8 and Figure 9 below. For an arbitrary scene, we would like to specify spawn locations and obstacle locations for our crowd scene using semantic labels. For each asset in the scene, it is possible to create an associated JSON file with default parameters shown in the schema in Figure 8. The user can then update the file to specify whether the asset should be treated as a spawn location, an obstacle or undefined. This can easily be done programmatically. An example of a configured JSON file corresponding to this schema is shown in Figure 9. This data can trivially be used for any other platform by simply reading the JSON file. When running the crowd simulation, we can parse the scene for all the assets with the "Spawner" label and then read in their corresponding spawn location. This is a simple example, but it demonstrates how this approach facilitates a consistent way to format data which can be used for any platform. This means that if the same scene were to be implemented using Houdini, for example, the scripts to actually use the spawn locations and obstacles would have to be re-written, but the interface remains consistent so the data does not need to be transformed or modified from the json file, simply read.



```
{
  "type": "object",
  "properties": {
    "AssetName": {
      "type": [
        "string",
        "null"
      ]
    }, "ObjectType": {
      "type": [
        "string",
        "null"
      ], "enum": [
        "Spawner",
        "Obstacle",
        "Undefined"
      ]
    }, "xPosition": {
      "type": "float"
    }, "yPosition": {
      "type": "float"
    }, "zPosition": {
      "type": "float"
    }
  }, "required": [
    "AssetName",
    "ObjectType",
    "xPosition",
    "yPosition",
    "zPosition",
  ]
}
```

Figure 8 An example of a schema for annotating



```
{
    "AssetName": "SpawnLocationOne",
    "ObjectType": "Spawner",
    "xPosition": 23.34,
    "yPosition": 0,
    "zPosition": 32
}
```

Figure 9 An example of a json file with semantic data objects as spawn locations or obstacles

4.3.2 Semantic Animation Search User interface

A main theme of SAUCE research is the development of semantic descriptors to promote the re-use of assets. In line with this, work was published by researchers at TCD [15] which is able to accurately and automatically classify animation data according to four classes: "bending down", "jumping", "running" and "walking". SAUCE deliverables 7.1 and 7.2 outline a detailed system for semantically annotating and storing numerous modes of data, including animation, which is actively used by multiple SAUCE partners. As part of our contribution, we have designed a lightweight standalone database querying tool that can query semantic labels and conveniently export animations to Unity.

We anticipate that for the sake of prototyping, the viewer and exporter will offer a significant time saving. For demonstration purposes, we have coupled this tool with animations from databases that have been liberally licenced for research purposes: the CMU Graphics Lab Motion Capture Database and Filmakademie's PHS motion library. Both databases contain data that has already been semantically labelled and the CMU library has additional semantic labels from the classifier in [15]. The viewer allows for the addition of arbitrary new animation sources. This provides an interface from the animation database to the Crowd Agent Manager, highlighted in red in Figure 10.

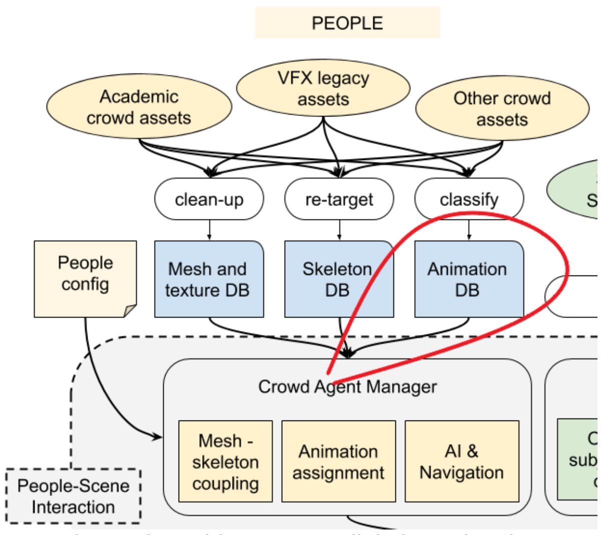
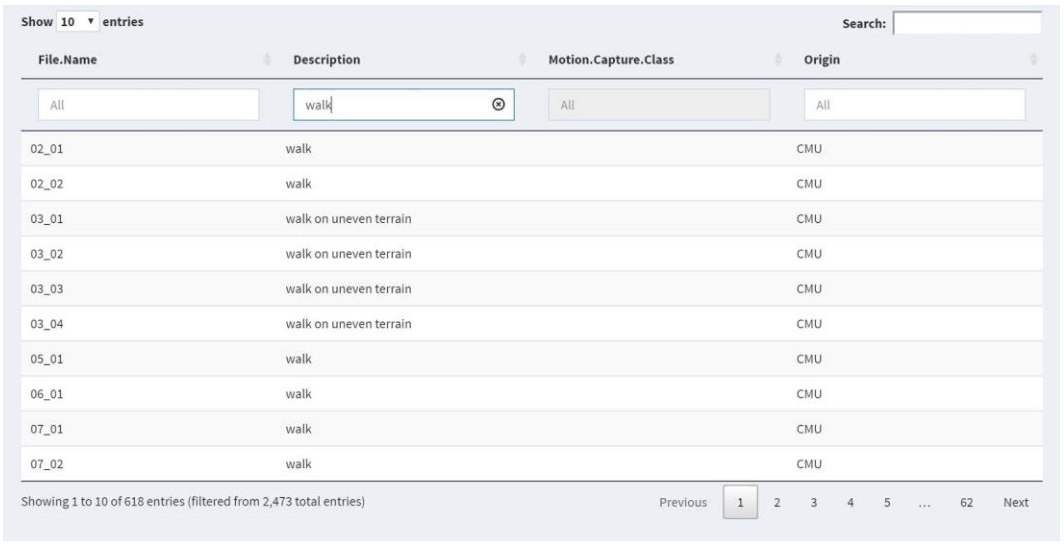


Figure 10 The Animation DB querying tool provides a way to link the Animation DB to the Crowd Agent Manager.

The database viewer was written with small animation studios in mind. While designing the viewer, we considered that these studios may not have well-designed databases to structure their data but may use a simple directory system to categorise their assets. This led to the decision to assume that the data is stored in a local file system. The semantic data is assumed to be held in a csv file, which also contains metadata such as the origin of the asset and licencing restrictions. The idea is that files can be uploaded to the file system with all their associated data and metadata in the csv file, which effectively acts as an indexing system. The upload and deletion of files to the local system with associated metadata in the csv file can easily be scripted.

The viewer assumes that the file metadata exists in a tabular format in the csv file - there are no restrictions on the number of fields, but the file name is mandatory. This is analogous to a primary key in a standard database system. The user can search each field present in the csv file. Animations in the table can be visualised (Figure 12), with the caveat that they must either be in Biovision .bvh format. The viewer was written using the R Shiny web development framework, which also means it can easily be configured to run on a server.

Figure 11 shows the sample search fields for the user interface. The user can create filters for each of these search fields. The figure demonstrates the user applying a filter for animations with a description that matches or partially matches "walk". Once they have applied the necessary filters, they can visualise animations to ensure they are suitable for their application. Figure 12 shows a sample frame of a "walk" animation from the CMU database. Finally, once the user is satisfied with the animations, they can add them to an export table, shown in Figure 13. This export table records all animations to be downloaded. Once the user presses the "download selected" button in Figure 13, the animations in this table are retrieved from the file system, zipped and made available in the users "downloads" folder, which can then typically be directly imported as a batch into the users simulation engine (Unity in our case).


Show 10 entries

Search:

File.Name	Description	Motion.Capture.Class	Origin
All	walk	All	All
02_01	walk		CMU
02_02	walk		CMU
03_01	walk on uneven terrain		CMU
03_02	walk on uneven terrain		CMU
03_03	walk on uneven terrain		CMU
03_04	walk on uneven terrain		CMU
05_01	walk		CMU
06_01	walk		CMU
07_01	walk		CMU
07_02	walk		CMU

Showing 1 to 10 of 618 entries (filtered from 2,473 total entries)

Previous 1 2 3 4 5 … 62 Next

Figure 11 Each metadata field can be searched.

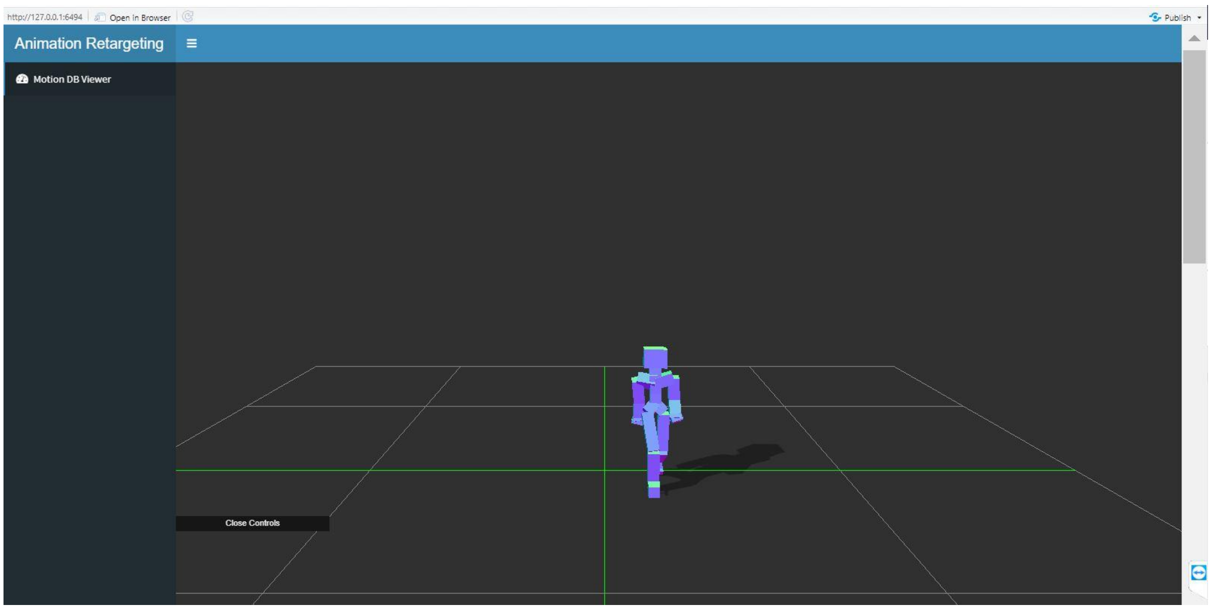
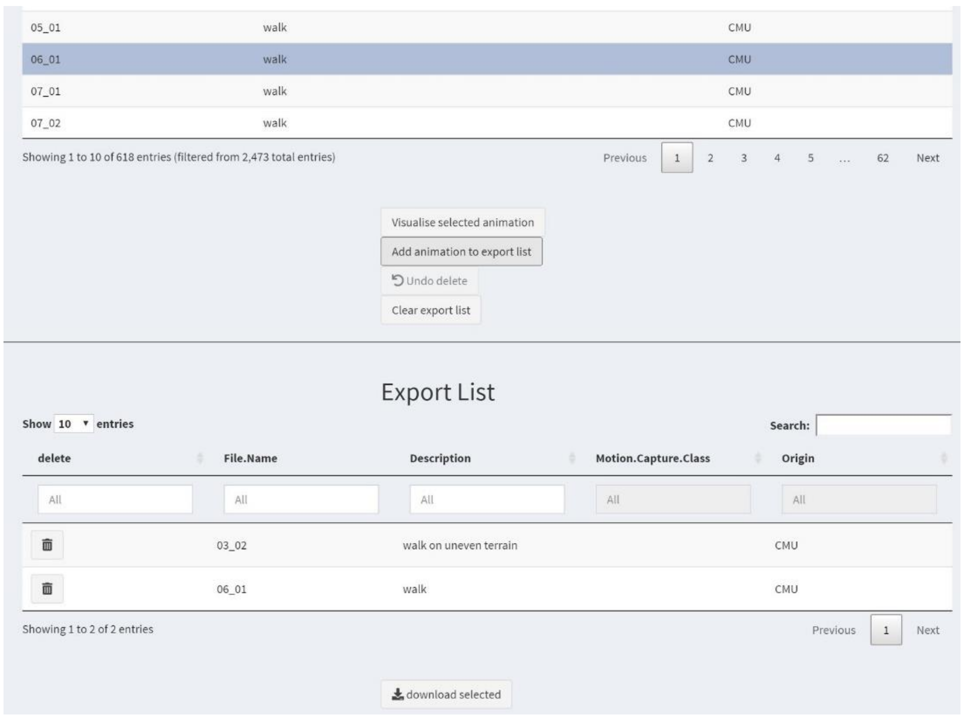


Figure 12 Animations can be viewed inline.



05_01	walk	CMU
06_01	walk	CMU
07_01	walk	CMU
07_02	walk	CMU

Showing 1 to 10 of 618 entries (filtered from 2,473 total entries)

Export List

Show 10 entries Search:

delete	File.Name	Description	Motion.Capture.Class	Origin
All	All	All	All	All
	03_02	walk on uneven terrain		CMU
	06_01	walk		CMU

Showing 1 to 2 of 2 entries

Figure 13 Animations can be added to an export list which can be directly imported to Unity or any other simulation engine.

4.4 Artificial Intelligence for Crowd Simulation

We implemented a number of Artificial Intelligence modules as part of our framework. These provide the logic to imbue crowd members with believable characteristics. These include:

- A standalone module to calculate calibrated paths to a goal destination around obstacles.
- A module which implements local avoidance among crowd members.
- A module which triggers animations among crowd agents based on semantic labels.
- A module which can stylize animations, which uses work completed by UPF in D6.6 Motion Stylization Implementation.

We provide the implementation details of these modules in the next sub-section. The Use Cases presented in Section 4.5 provides tangible examples of how these modules can be used to rapidly prototype crowd scenes.

4.4.1 Potential Fields for High-Level Navigation

High-level navigation here refers to the calculation of a viable path from one point to another, taking into account soft constraints. It does not directly consider artefacts such as potential collisions, which are treated separately as a post-processing step. The criteria for high-level navigation are derived from the objectives and goals identified in Section 3.1:

1. The system must perform in real-time.
2. The system must provide autonomy: excessive manual intervention is not desired.
3. The system must provide adequate functionality to allow real-time prototyping.

We evaluated a number of approaches based on existing literature and chose the potential field approach as the most suitable based on the identified criteria. The potential field implementations for crowd simulation outlined in [16] and [17] provide a basis for our own implementation.

The implementation of the potential field module was written as a standalone application in python, due to its abundance of scientific libraries and rapid prototyping time. The language-agnostic details are presented here. For an in-depth exposition on potential fields in AI, we refer the reader to Chapter

4 of [16], which our implementation is based on. Modifications made to reflect the differences in simulated human motion to robot motion.

4.4.1.1 Specifying the Potential Function

A potential field can be represented as a function of spatial coordinates $U : \mathbb{R}^m \rightarrow \mathbb{R}$ that is differentiable. We are concerned only with the two-dimensional case for crowds: $U : \mathbb{R}^2 \rightarrow \mathbb{R}$. The potential can be thought of as energy, with the gradient interpreted as force, meaning that each crowd agent can use the potential field to calculate a force to be applied to it.

The potential at any given point in $\mathbb{R}^2$ is calculated as the sum of an arbitrary number of differentiable functions of the form $U_i : \mathbb{R}^2 \rightarrow \mathbb{R}$, i.e. $U = \sum U_i$. This allows the potential function to be configured to incorporate 'hills' and 'troughs', which affects how a crowd member moves in the field. Given this potential function, each crowd member can follow a path 'downhill' by following the negated gradient of the potential function. Figure 14 illustrates an example of a potential field in 3 dimensions, while Figure 15 shows the projection of this surface onto 2 dimensions in the form of a contour plot.

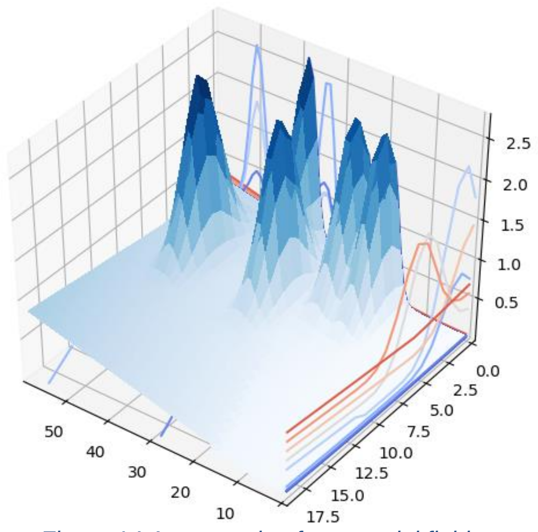


Figure 14 An example of a potential field.

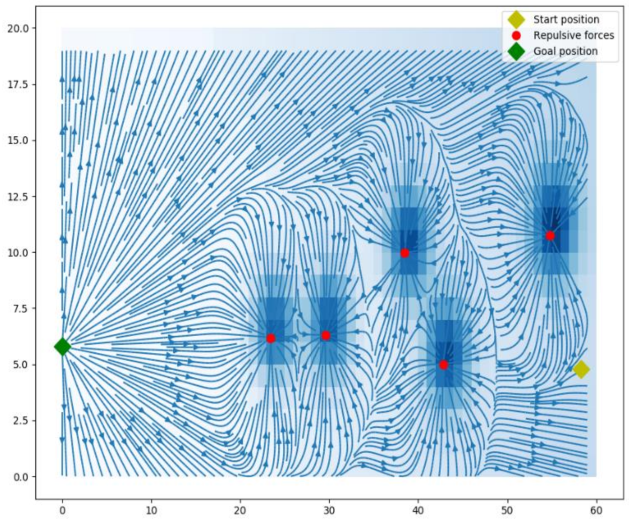


Figure 15 2D projection of potential field with contour plot

The end-user can specify each U_i to be used in the potential field, however this is typically a highly tedious. We provide 2-D Gaussian distributions to specify the U_i, since it is very easy to specify the scale, location and shape of these functions. Figure 16 and Figure 17 show how the covariance matrix can be narrowed along each axis, which means they can effectively approximate different shapes. Note that Figure 16 can approximate a rectangle when projected onto two dimensions, whereas Figure 17 approximates a circle. Gaussian distributions also have the advantage of having a strong local effect but a weak global effect. This means that they can be defined to intuitively specify the agent's path locally.

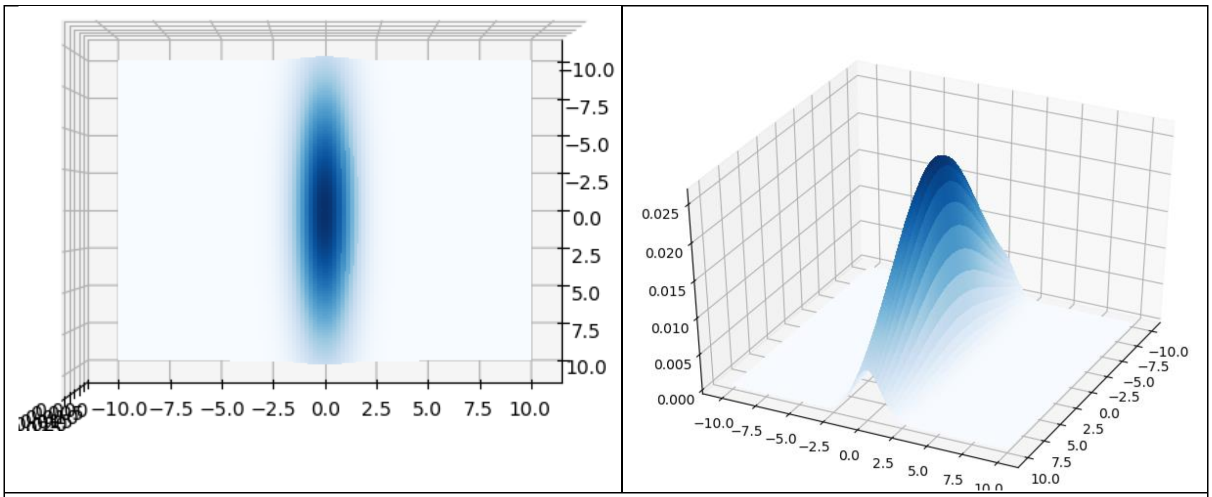


Figure 16 Gaussian with covariance matrix $\begin{pmatrix} 6 & 0 \\ 0 & 1 \end{pmatrix}$

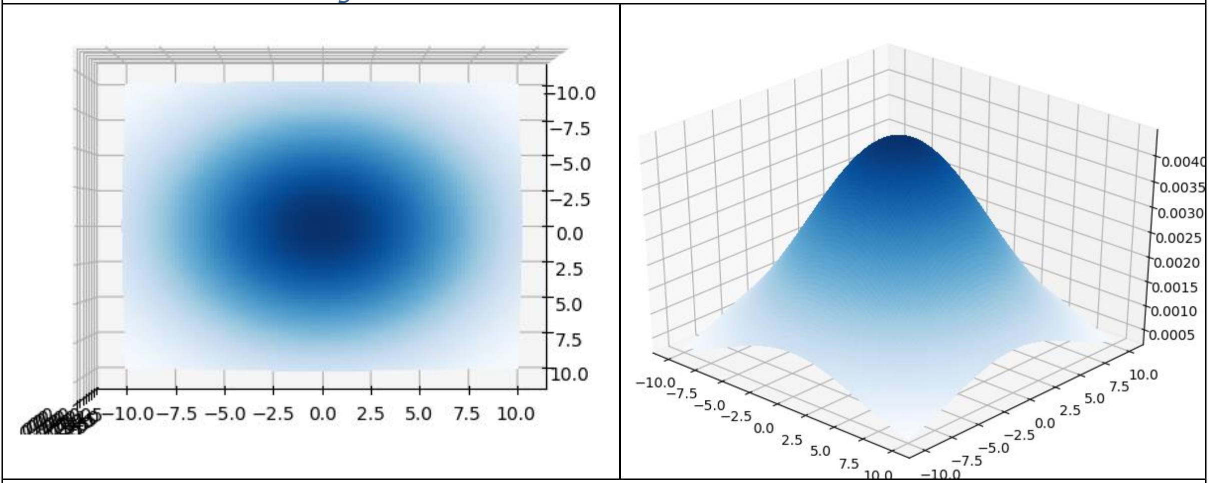


Figure 17 Gaussian with covariance matrix $\begin{pmatrix} 6 & 0 \\ 0 & 6 \end{pmatrix}$

Given an arbitrary potential function, $U = \sum U_i$, a gradient descent algorithm can be applied to derive a path, as noted above. Once again, we refer the reader to [16] for details in this context. However, there are a number of drawbacks to "vanilla" gradient descent:

1. It may not succeed in finding a path to the goal if local minima are present in the potential function.
2. If there is a slight "ridge", it may not succeed in finding a path to the goal.
3. Gradient descent requires a step-size which is small enough to avoid skipping over ridges and minima/maxima, but not so small as to cause excessive computation time.

Common remedies to these issues are to ensure there is only a single global minima, to discretize space and to incorporate backtracking/random walks. Since we do not intend to apply this algorithm for true navigation in an unknown scenario, we instead propose the following:

- The main objective of crowd simulation is to create a group of "seemingly sentient beings".
- The chief concern for high-level navigation is therefore to create a *believable* path.
- *Believability* is a subjective measure; therefore, we must provide a highly configurable path instead of a single "correct" path. If the user is not satisfied with the calculated path, they should be able to modify it until it is believable.
- The agents must be able to use the calculated paths in real-time.

- The paths should provide a realistic and smooth trajectory to satisfy the *believability* criteria.

Based on these principles, we modified the vanilla gradient descent algorithm so that for a "reasonable" input, a path is usually successfully calculated. This path can be easily configured by altering the potential function using gaussian functions. The key modifications to vanilla gradient descent to accommodate this are:

- The addition of a momentum term to alleviate the local minima issues.
- The addition of a term to guide the crowd member in the direction of the goal, in case the gradient guides the crowd member away from the goal.
- A post-processing step to calculate a piecewise linear function that reduces the number of points on the trajectory, providing a motion that does not oscillate.

The algorithm for calculating the path using a modified gradient descent is shown in Figure 18.

```
Algorithm 1 Modified Gradient Descent for Path-finding
Input: A means to compute the gradient $\nabla U(q)$ at a point q $\in$ R$^2$.
    $\alpha$, a coefficient for the momentum term.
    $\beta$, a coefficient for the goal attraction term.
Output: A sequence of points that make up the agent's path

1:  $\vec{momentum} \leftarrow 0$
2:  $q_0 \leftarrow q_{start}$
3:  $i \leftarrow 0$
4:  $t_0 \leftarrow time$
5:  while $time < maxtime$ and $\varepsilon < \|q_i - q_{goal}\|$ do
6:      gradient $\leftarrow -\nabla$U(q$_i$)
7:      vector to goal $\leftarrow goal - q_i$
8:      $momentum \leftarrow \alpha \times momentum + (1-\alpha) \times gradient$
9:      $v_i \leftarrow \alpha \times momentum + \beta \times vectortogoal + \gamma \times gradient$
10:     $q_i \leftarrow q_{i-1} +$ step size $\times v_i$
11: end while
12: return $q_0, \ldots, q_i$
```

Figure 18 A modified Gradient Descent for path-finding with a potential field

4.4.1.2 Standalone Executable Interface Specification

We implemented this algorithm as a standalone executable. The interface requires the following input parameters:

Parameter Name	**Description**
Goal Position	The (x,y) location of the goal position.
Goal Potential Coefficient	A coefficient which scales the potential of the goal position (higher values mean stronger attraction).
Start Position	The (x,y) location of the start position.

Repulsor Locations	The (x,y) positions of the Gaussian repulsors .
Repulsor Standard Deviations	The covariance matrix of each repulsor (in 1-1 correspondence with Repulsor Locations) representing the 2D "spread" of each repulsor.
Repulsor Potential Coefficients	The coefficients used to scale the calculated potential of the Gaussian repulsors.
Attractor Locations	The (x,y) positions of the Gaussian attractors.
Attractor Standard Deviations	The covariance matrix of each attractor (in 1-1 correspondence with Attractor Locations) representing the 2D "spread" of each repulsor.
Attractor Potential Coefficients	The coefficients used to scale the calculated potential of the Gaussian attractor.
Timeout	The maximum amount of time that can be used to find a path to the goal.

The output is provided as a sequence of (x,y) points, leading from *Start Position* to *Goal Position*. A brief analysis of the performance of this standalone module is provided in Table 6.

4.4.1.3 Waypoint Length Reduction via Post-Processing

Once a suitable path for each crowd member has been calculated, they need to be able to calculate suitable velocities to traverse this path. Velocity updates can be disjointed if the points on the path are close together. We developed a waypoint length reduction algorithm to address this problem.

The reduction of the number of waypoints for the calculated path is done via a post-processing step. This takes the sequence of (x,y) points and applies the following procedure:

1. A piecewise linear model is desired to fit the data piecewise, for *m*+1 segments. In order to fit the model, *m* breakpoints are required. There is a trade-off between the number of breakpoints, *m*, and the quality of the fit, measured by the Residual Sum of Squares (RSS). We allow the user to specify *m* as a proportion of the original number of points.
2. We find the breakpoints using the R *breakpoints* function, found in the strucchange library [18] which is an implementation of the method found in [19]. This method is based on a dynamic programming approach and provides an estimate for the vector of breakpoints *m*.
3. We then use these breakpoints to fit a piecewise linear model, which uses the breakpoints found in 2. This is done using the R segmented function from the segmented library.
4. Finally, we use the linear model from 3 to provide estimates of the dependent variable at the breakpoints.

A result of the application of this algorithm can be seen in Figure 19.

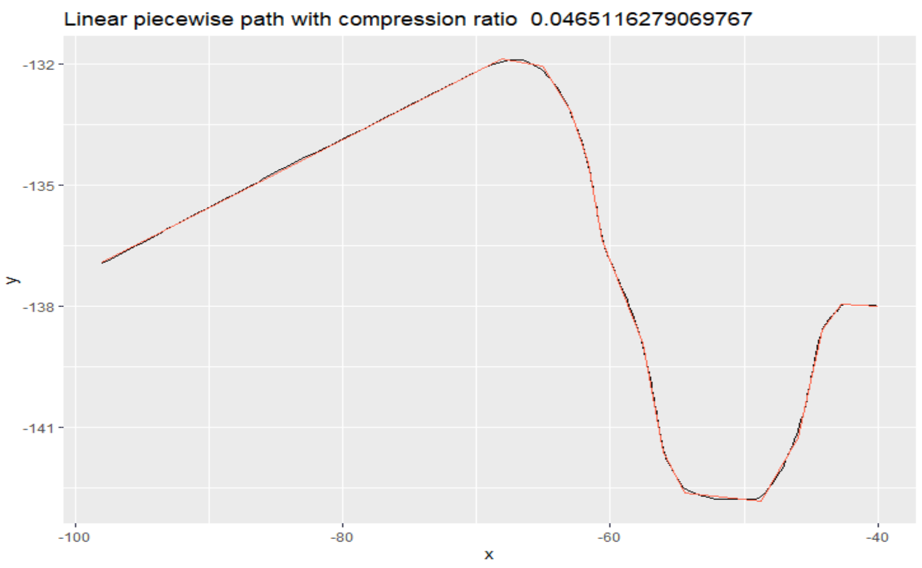


Figure 19 The black line represents (x,y) points outputted by modified gradient descent. The red line represents the corresponding (x,y) points calculated by the piecewise linear regression, decreasing the number of (x,y) points on the path by a factor of ~20

4.4.1.4 Interface With Semantic Data

Section 4.3.1 outlines how we intend for semantic data to be used to drive the crowd behaviour. Instead of asking the user to specify every single parameter for the potential field path calculation, we created an interface between the path planning module and the semantic data annotation module. Specifically, we provide the user with a schema, which can be attached to scene assets. Once this schema has been used to create a JSON file, the path planning module can then either infer or directly read all the input values from this file.

This has three key merits:

1. It offers a significant time saving to the user by avoiding the manual specification of all parameters.
2. It guarantees that the user has specified the data in the right format.
3. It guarantees consistency in how the data should be formatted - if there is an issue with the input data, this will be detected before the path calculation begins.
4. It promotes re-use: the schema and path-finding module can be re-used across scenes.

The first is obvious but the other points are more subtle. Generally speaking, AI modules require a specific input in order to calculate an output. A frequent source of error is inputting data that is incorrectly in format or context: this is often summarized with the phrase "Garbage in, garbage out". In order to avoid occurrences of this, we first validate any data that will be fed into the pathfinding module using the JSON schemas explained in Section 4.3.1.

This was only partially carried out in Unity due to implementation complexities. The JSON schema is intended to reflect the user configurable input parameters. Future iterations of this work could build on this idea to develop a mature and robust interface.

4.4.1.5 Potential Field Goal Chaining

An obvious limitation of the above approach is that a path to an only single goal location can be calculated in advance. We anticipate that the user may know in advance that the crowd agent should move between sequential goal points. We offer a simple extension as follows:

- The user may set out several goal points that are "chained" - they are assumed to be in sequential order.

- Specifically, we let the user specify a set of triples: <start_point$_i$, potential_field$_i$, goal_point$_i$>. Each of the triples in the set is used to calculate a path from start_point$_i$ to goal_point$_i$ using potential_field$_i$. It is required that start_point$_{i+1}$ is equal to goal_point$_i$, to ensure continuity.

This is implemented by iteratively applying the path calculation (implemented as a standalone executable) to each triple, and then simply combining each of these paths together. The result of this can be seen in Figure 20, where the three sequential goal locations are shown in green and the repulsive forces are highlighted in purple. The calculated path is shown in yellow. The accompanying set of triples used in the configuration is shown in Figure 21.

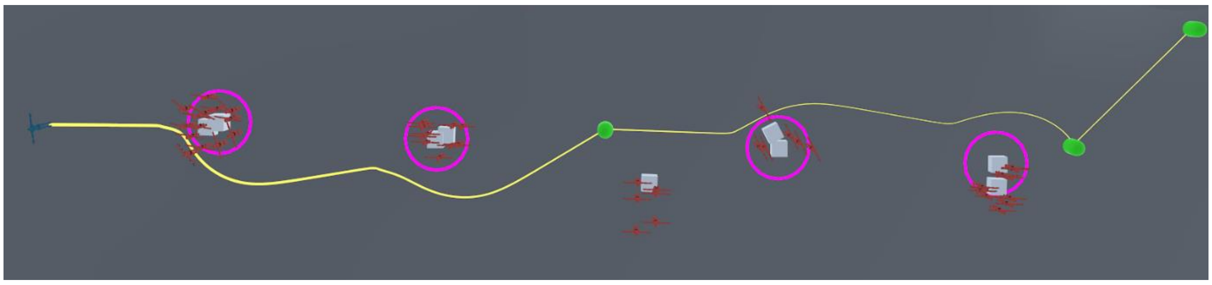
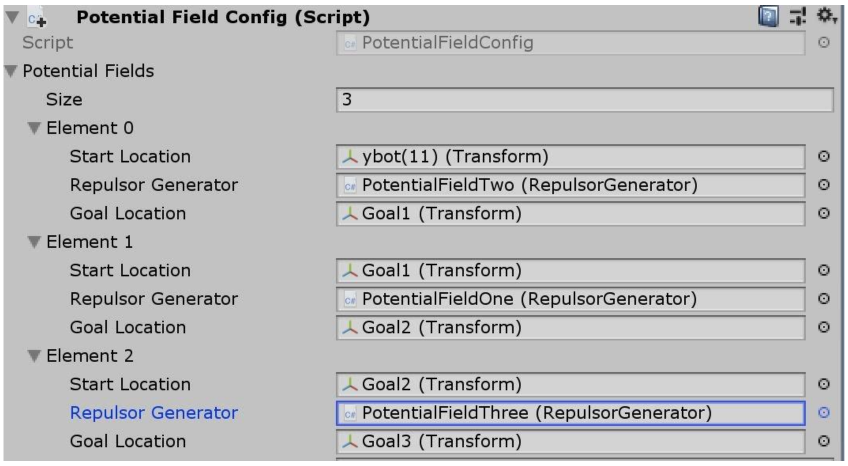


Figure 20 Goal locations are shown in green. Locations of repulsive forces are shown in purple.


Potential Field Config (Script)

Script	PotentialFieldConfig
▼ Potential Fields	
Size	3
▼ Element 0	
Start Location	ybot(11) (Transform)
Repulsor Generator	PotentialFieldTwo (RepulsorGenerator)
Goal Location	Goal1 (Transform)
▼ Element 1	
Start Location	Goal1 (Transform)
Repulsor Generator	PotentialFieldOne (RepulsorGenerator)
Goal Location	Goal2 (Transform)
▼ Element 2	
Start Location	Goal2 (Transform)
Repulsor Generator	PotentialFieldThree (RepulsorGenerator)
Goal Location	Goal3 (Transform)

Figure 21 The configuration used to calculate the path displayed in Figure 20. Note that start_point$_{i+1}$ is equal to goal_point$_i$

4.4.1.6 Waypoint Editing

As mentioned throughout this deliverable in relation to path planning, a key aspect for our target end users is believability. To that end, we implemented a waypoint editing tool which allows the user to manually specify the positions of cached waypoints calculated by the potential field module if they are not fully satisfied with the automated results. Once the user has finished editing the waypoints in real-time, they can save their updated positions to the cache. This allow real-time editing of the automatically generated path through the Unity interface. We implemented this in Unity by displaying the transforms of each waypoint using a "capsule" object, allowing the user to drag each to the desired position. This is demonstrated in Figure 22 - Figure 25, where edit mode is entered in Figure 23, the pre-calculated path is updated in real time in Figure 24 and then edit mode is exited and the updated path is saved in Figure 25.

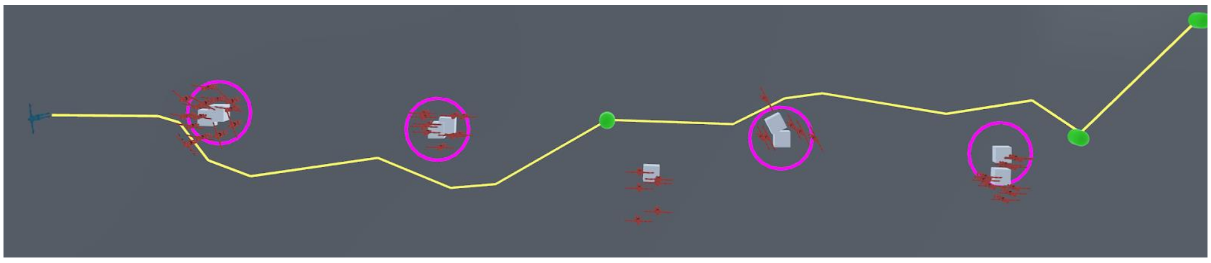


Figure 22 Automated planned path displayed.

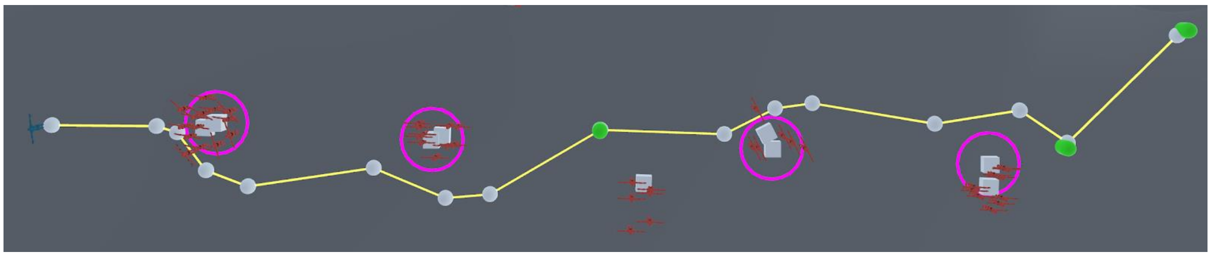


Figure 23 Edit mode engaged

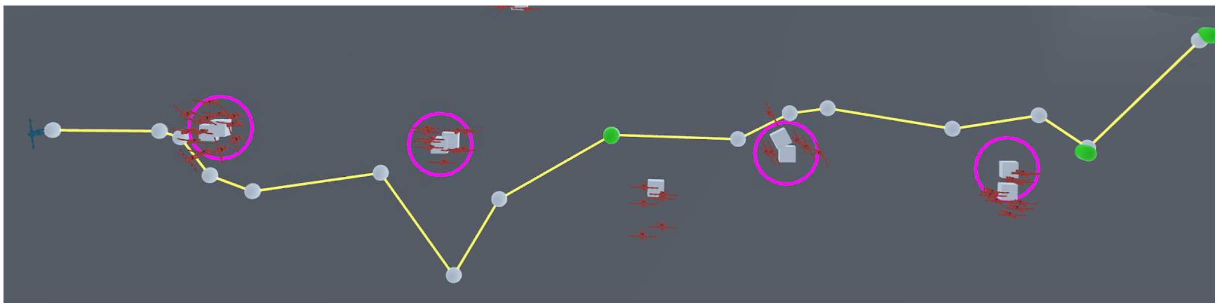


Figure 24 Waypoint dragged to new position

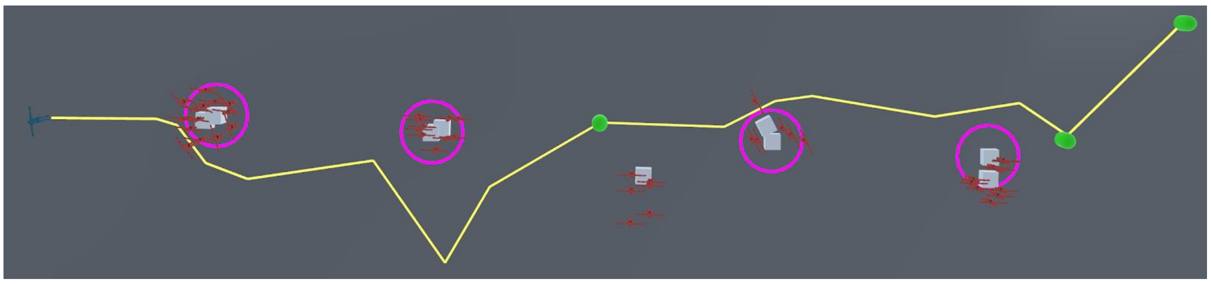


Figure 25 Updated path saved to cache and edit mode exited.

4.4.1.7 Unity Implementation

The complete integration of this module in Unity allows for paths to be automatically calculated **in advance** of running the simulation in Unity. We had the choice to either integrate this module to perform the calculations in real-time, or to carry them out in advance of the simulation run. We consciously chose to carry out the calculations in advance of the simulation run, for the following reasons:

- Crowd scenes typically require a lot of resources for rendering and updating, so cutting down on any run-time computation is advantageous.
- Calculating paths before run-time means that the user can check that they are suitable in advance of running the simulation. We implemented a real-time way-point editing tool, outlined in Section 4.4.1.6, to allow the user to update and save automatically calculated waypoints before run-time.

Once the crowd members configuration has been set in Unity, the configuration is read and the paths are automatically calculated and cached when the user clicks the "write path to file for selected agents" button in the navigation toolbar. This must be done **in advance** of running the simulation. If the user updates the configuration by manually moving around crowd members and repulsive forces, they will need to re-calculate the path by clicking the "write path to file for selected agents" button in the navigation toolbar again. Once the simulation is running, the cached paths for each crowd member are read and followed. They cannot be adjusted while the simulation is running.

4.4.2 Local Avoidance Implementation

The potential field path planning in Section 4.4.1 does not account for the fact that calculated paths may lead to conflicts – crowd members may collide while traversing. This problem is usually resolved by implementing a post-processing "local avoidance" algorithm. This has a large body of literature behind it and many mature, robust and efficient libraries have been implemented.

Unity's in-built Reciprocal Velocity Obstacles (RVO) library provides an implementation of local avoidance [20] . It does so by auto-assigning a collision avoidance radius and "avoidance priorities". A higher priority means the associated game object will maintain its traversal speed to accommodate smooth navigation, while game objects with lower priorities "give way" and alter their traversal speeds to prevent collisions.

This occasionally gives rise to an outlier scenario. The crowd members may undergo oscillations of their velocities [21]. If two crowd members are assumed to travel with velocities v_A and v_B respectively and possess overlapping paths, in order to avoid potential collisions, either crowd member A or crowd member B alters their velocities to v'_A or v'_B to let the other member pass. As their old velocities lead them directly to their goal positions, each then revert to their previous velocity. This process continues until either of the members have reached their destinations, leading to repeated oscillations in their old and new velocities. Due to the random sampling nature of RVO, this application of local avoidance sometimes causes the members to lock onto each other's collision avoidance radius and not let the other pass, termed as a "reciprocal dance". This issue prominently arises in high density of crowd members, specifically in regions with narrow passages due to the crowd members having a limited amount of space to navigate around each other and the velocities of each agent in proximity affecting local avoidance as highlighted in Figure 26.

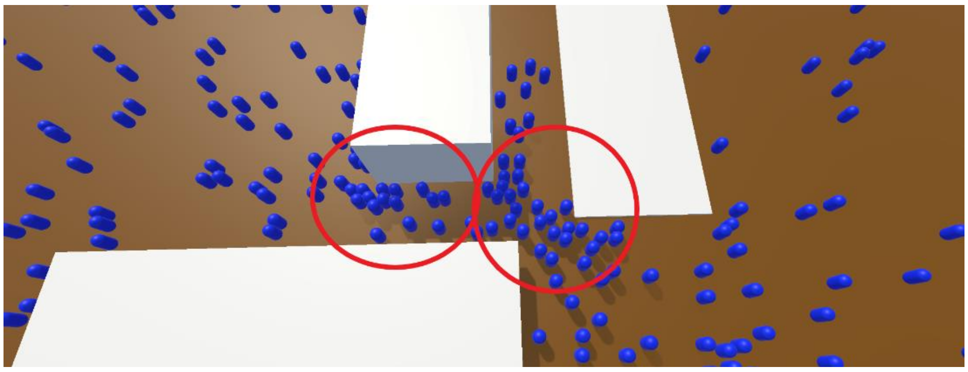


Figure 26 Example scenario that may lead to reciprocal dances

This deadlock can be overcome by using the Reciprocal Velocity Obstacles 2 (RVO2) library [22], which is an iteration of the Reciprocal Velocity Obstacles (RVO) library [21]. RVO2 uses Optimal Reciprocal Collision Avoidance (ORCA) for local avoidance. It considers a global radius in the entire scene and keeps track of the agent's positions and velocities. This solution determines the agents that are in vicinity to one another globally and causes them to appropriately slow down. Under the library's documentation [23], RVO2's "ORCA defines velocity constraints with respect to other agents as half-planes, and an optimal velocity is efficiently found using (two-dimensional) linear programming. In contrast, RVO Library 1.x uses random sampling to find a good velocity". Figure 27 demonstrates a simple scene using the RVO2 library.

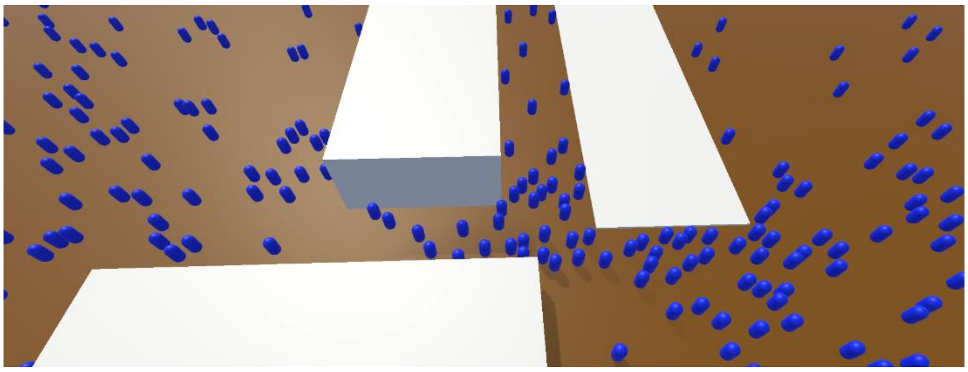


Figure 27 No deadlock scenarios observed

Unity's NavMesh Agent provides rudimentary components for navigation, namely, A* pathfinding and dynamic obstacle avoidance. These components are by default linked to the RVO component, that takes in the agent's velocity to determine ideal local avoidance. Our implementation also allows an RVO2 component to be attached to prevent the reciprocal dances and deadlock caused by the RVO library. Although the integration of RVO2 as an addon may impact performance due to basic calculations being performed by the in-built RVO and then overwritten by the RVO2 library, the RVO2 library provides a typically better local collision avoidance and we found the trade-off to be worthwhile in scenarios involving large crowds. Details of the performance impact can be found in Table 3.

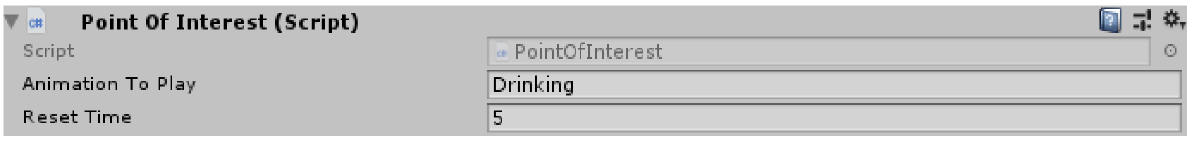
4.4.3 Integration of Animation Triggers

This sub-section reports a way to link semantic annotations in the scene with animations for crowd members, which provides a simple and scalable way to produce believable behaviour.

Since we are dealing with crowd scene with potentially thousands of members for a real-time application, evaluating complex behaviour trees or action utilities for each agent threatens a significant drop in framerate. Although these approaches offer a high level of individuality and autonomy, which were identified as desirable traits in 8, they also often require extensive configuration which is not suitable for prototyping scenes with large crowds. In future iterations, we would ideally incorporate these approaches in an automated way for large scale crowds. Our chosen approach builds on the semantic theme of SAUCE. We describe the general steps and our Unity-specific implementation.

Our framework allows animations for agents to be triggered via spatial cues. We detect when an agent has entered a specific region. This region has an associated semantic action – for example the region in front of a water fountain could have an association semantic "drink" action. Once the agent enters this region, if a "drink" animation is present among the set of animations it will pause the agents navigation component, play the animation and then resume the agents navigation component. The major advantage of this approach is that it is one-to-many: one semantic drink animation can trigger many different "drink" animations for different crowd members. We rely on semantic labelling for this to be carried out. A further advantage to this is that it allows agent to query semantic labels and then decide on appropriate behaviour. For example, if an agent was created with a utility-based artificial intelligence to make decisions, a high utility value for actions that result in an environment state with low "thirst" should lead to a decision to "drink". This action can be performed by navigating to an appropriate semantically annotated environment asset.

In the Unity game engine, we implemented this using physics collider. Once the collider is placed in the scene, we associate it with a semantic animation using a string label. This is shown in Figure 28.


Point Of Interest (Script)

Script	PointOfInterest
Animation To Play	Drinking
Reset Time	5

Figure 28 Semantic description provided to a point of interest

Colliders are attached to agents which will trigger the animations. Unity engine lets crowd members play animations through animation controllers [24], which can be accessed through a script. An example of this can be found in Figure 29, which shows an animation controller that plays a walking animation by default and can transition to a drinking animation. The script matches the semantic label in the collider against the list of names of animations for the crowd member that has triggered the collider. If an animation is successfully matched, the agents navigation component is paused, the animation plays and then the agent resumes navigation, illustrated in a bare-bones Unity scene in Figure 30. In this image, the crowd members collider intersects with the "drink" collider, which triggers the drink animation to play, with a water fountain placed by the collider for illustration purposes.

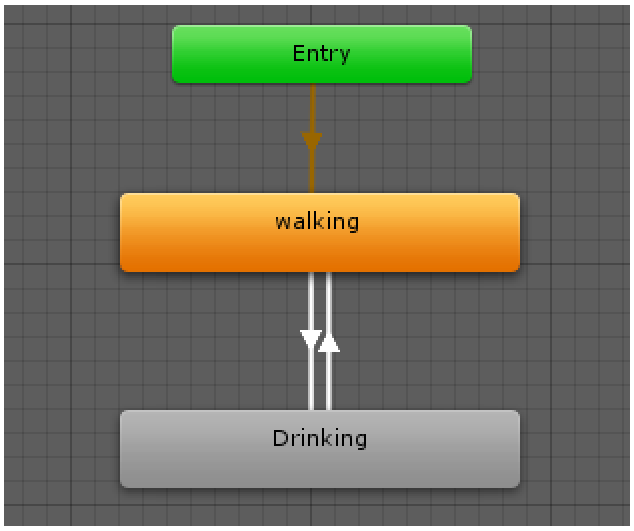


Figure 29 Animation triggered on Unity-based physics collision detection

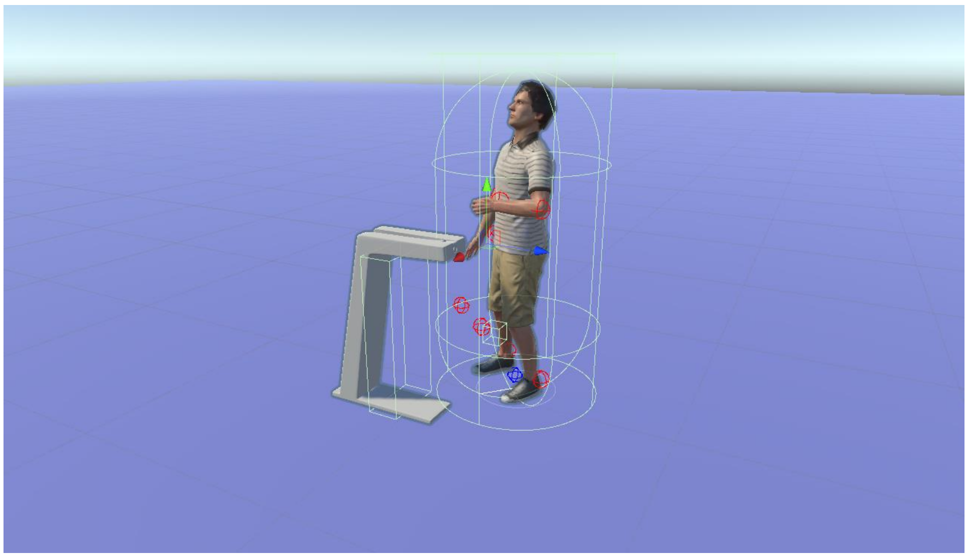


Figure 30 Animation triggered using Unity-based physics collision detection

This simple approach allows us to provide characters with a basic understanding of their immediate environment using semantic labels.

4.4.4 Motion Stylization Implementation

The final addition to our AI toolset is a motion stylization module, which uses work completed by UPF in D6.6 Motion Stylization Implementation. D6.6 outlines a framework that was written using OpenGL which calculates post-processed stylized poses by modifying joint rotations. We refer the reader to D6.6 for details of the algorithm.

We helped UPF to create a C# version of their motion stylization module, making it available as a Unity component. The module is compatible with animations that can be imported in the Unity *humanoid* configuration [25], which is part of the built-in Unity *Mecanim* animation system [26]. The algorithm behaves in the same way to the WebGL implementation and the input and output parameters match the WebGL implementation. Figure 31 - Figure 33 show the results of applying this module to a walk animation in Unity, with the associated Unity interface in Figure 34.

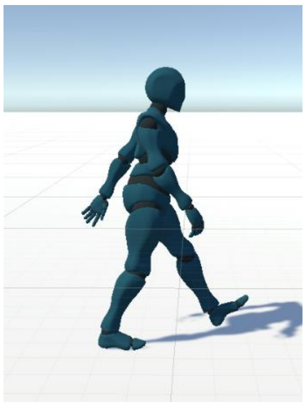


Figure 31 Neutral pose

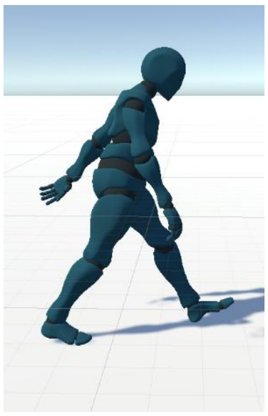


Figure 32 "Valence" value altered

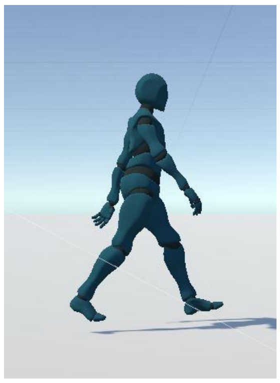
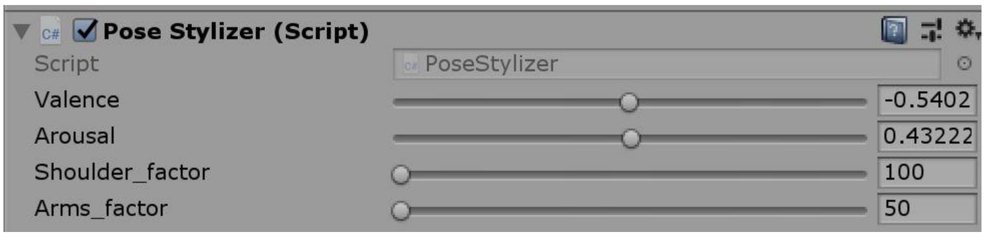


Figure 33 "Shoulder factor" altered


Pose Stylizer (Script)

Script	PoseStylizer
Valence	-0.5402
Arousal	0.43222
Shoulder_factor	100
Arms_factor	50

Figure 34 Unity Interface for Stylization Parameters

We incorporated this interface into our framework by linking it to the semantic labels present in the scene. This fits into our framework as part of the real-time (Unity) implementation, where behaviour is determined and implemented using semantic information, displayed in Figure 35. The relationship between the TCD modules and the UPF modules is displayed in Figure 36. We wrote a script that can read in semantic data from the structured sources documented in Section 4.3.1 and calculate estimated "mood" values based on the semantic data. The calculated "mood" parameters are then translated to their corresponding parameters in the UPF pose stylizer to create dynamically calculated stylistic animations.

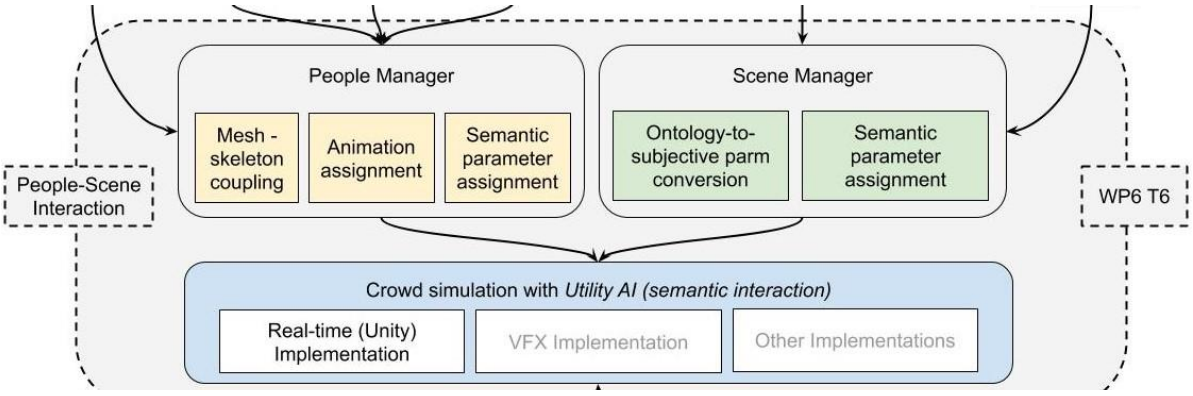
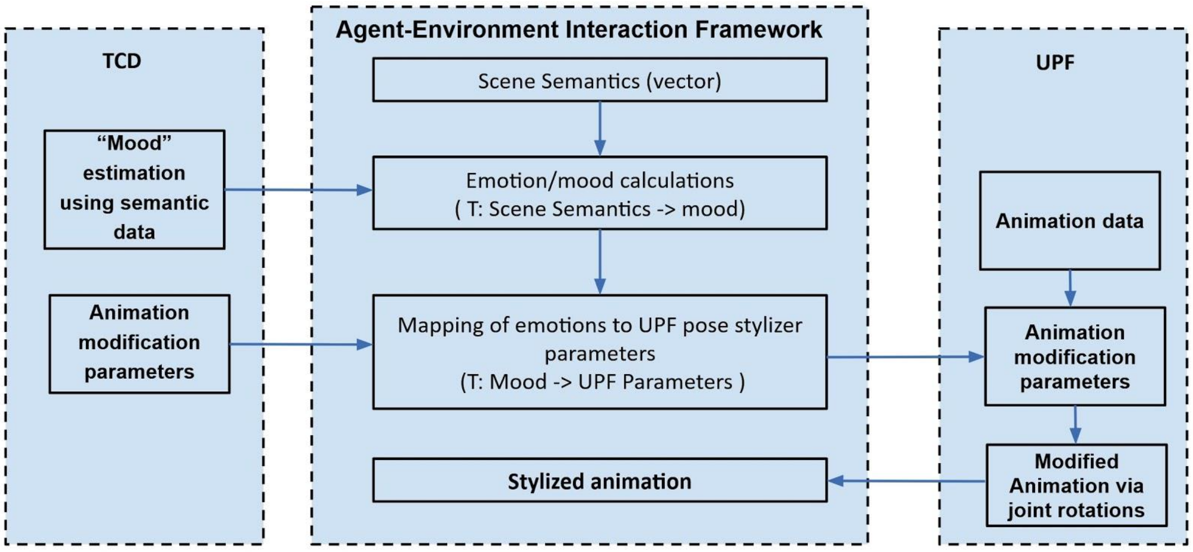

People Manager
Mesh - skeleton coupling
Animation assignment
Semantic parameter assignment
Scene Manager
Ontology-to-subjective parm conversion
Semantic parameter assignment
People-Scene Interaction
WP6 T6
Crowd simulation with Utility AI (semantic interaction)
Real-time (Unity) Implementation
VFX Implementation
Other Implementations

Figure 35 Stylistically modified animations are part of the real-time implementation


TCD
"Mood" estimation using semantic data
Animation modification parameters
Agent-Environment Interaction Framework
Scene Semantics (vector)
Emotion/mood calculations (T: Scene Semantics -> mood)
Mapping of emotions to UPF pose stylizer parameters (T: Mood -> UPF Parameters)
Stylized animation
UPF
Animation data
Animation modification parameters
Modified Animation via joint rotations
Figure 36 High-level flowchart showing how animations are stylized in the crowd simulation framework

4.4.5 Object Pooling for Runtime Creation and Destruction

This sub-section addresses runtime creation and destruction tools, which allow the user to create crowd members while the scenario is already running. This is useful in cases where the end user may want to create a never-ending "flow" of crowd members. Since there are a large number of components that could potentially be added to each crowd member, documented in subsections 4.4.1 - 4.4.4, the runtime creation of these members could lead to performance lags. This depends on Unity's garbage collector, which is responsible for allocation and deallocation of memory. Unity's memory management system uses a "stop-the-world" garbage collector, which pauses the program and then resumes following collection. This leads to potential lags in the execution of the code, explained in the Unity documentation [27]. "Incremental" garbage collection [27] can alleviate these lags by breaking down large garbage collection tasks into smaller micro-tasks. The execution of some of the smaller tasks can then be delayed until necessary, smoothing out performance drops. We found this method is also unsuitable for our task, since the creation and deletion of many crowd agents during run-time still led to large drops in frames per second due to a reciprocal large number of micro tasks being generated.

We implemented the object pooling design pattern for the creation of crowd members to alleviate these problems. Object pooling initialises crowd members in a "pool", created at the start of run-time. This ensures that any creation and deletion of members occurs without the intervention of the garbage collector, which is traded-off against additional memory use. Any crowd members removed from the running scene are returned to the pool, leading to improved performance.

For testing purposes, we spawned a crowd member with a simple mesh into a scene a varying number of times using the both standard approach for spawning and our Object Pooling implementation. The scene contained no geometry other than a ground plane, in order to record the frames per second of the scene with minimal interference from external factors. We compared of frames per second for crowd agents spawned through object pooling vs. the standard Unity technique, the outcome of which is shown in Table 2 in Section 4.6.1.2. The table clearly shows a boost in start-up performance with the object pooling implementation.

4.5 Use Cases

This sub-section describes three use cases, which demonstrate the capabilities of our crowd simulation framework as an integrated system. They were chosen to internally estimate the versatility of our system for realistic scenarios, significantly aiding development. We deliberately chose use-cases which reflect the cases frequently identified as desirable in the literature.

4.5.1 Social Distancing in an Open Square

The COVID-19 pandemic has created a situation where the phenomenon of *social distancing* is globally commonplace. Due to this, there is increased interest in designing spaces that can accommodate social distancing. We chose a social distancing scenario as a use case due to its relevance and the value of simulating this phenomenon without needing to involve real participants. This scene is comparable to other commonly modelled crowd scenes in open spaces, such as the Hajj pilgrimage [28].

We used an environment model taken from a previous project developed by TCD, named Metropolis [10]. This is a high-fidelity virtual representation of the TCD campus, with a rich and diverse set of assets. Figure 37 and Figure 38 show the environment model without any crowd members.

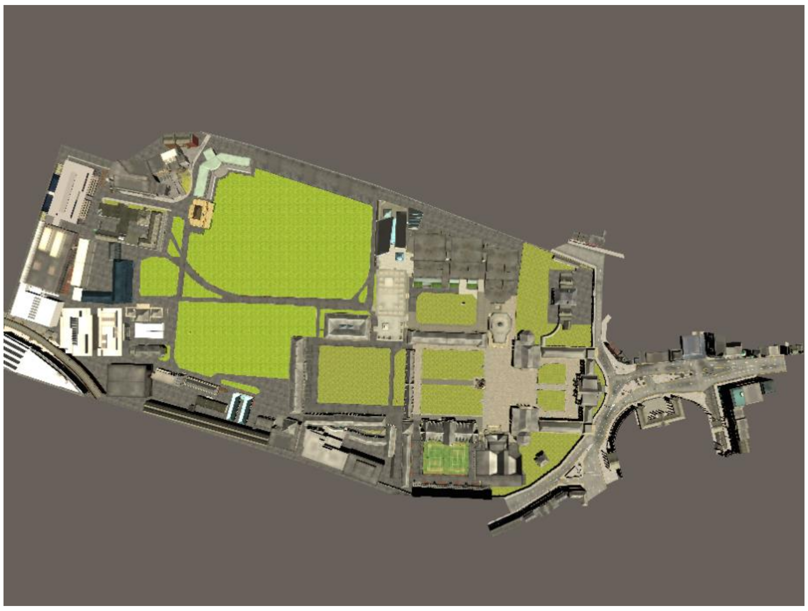


Figure 37 Top View of Metropolis

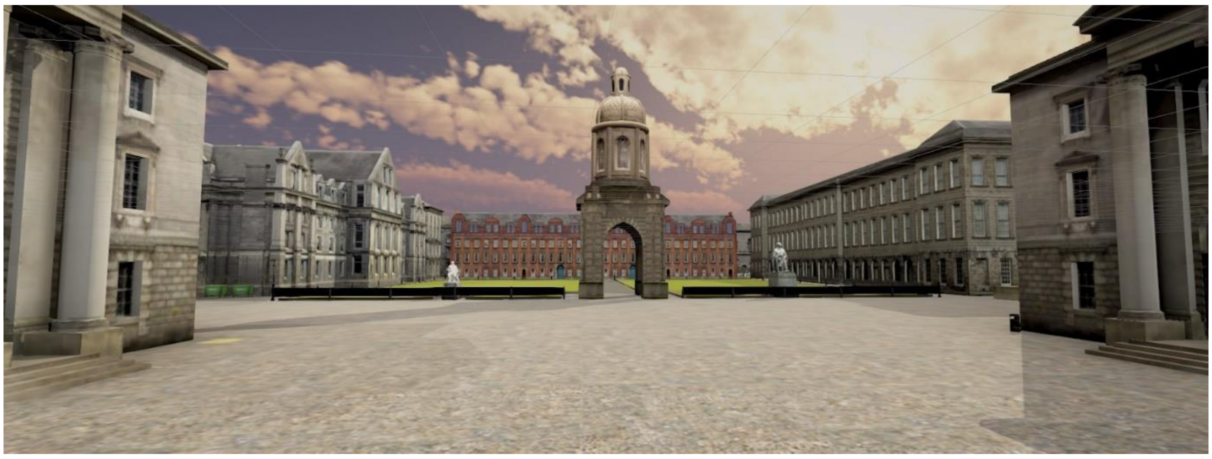


Figure 38 The main square in Metropolis

We provide a description of some of the steps taken to create the social distancing scene, which make use of the tools described in this document:

1. Semantic data for this environment was available from the DublinCity LiDAR dataset [29], [30]. We wrote a script to register the LiDAR model with the Metropolis model. We then exported the door

locations of the registered LiDAR model and created semantically annotated objects at the door positions which mark them as feasible start/spawn locations for crowd members. This process is shown in Figure 39 - Figure 41.

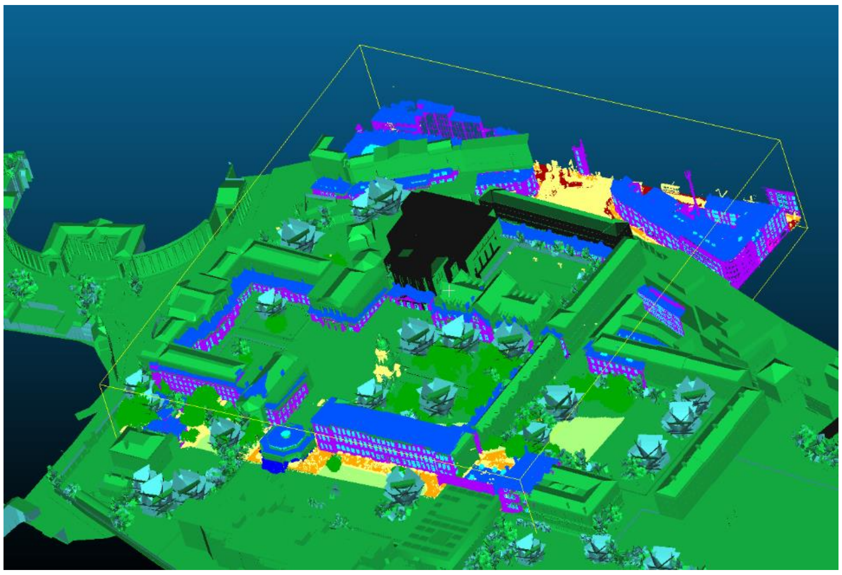
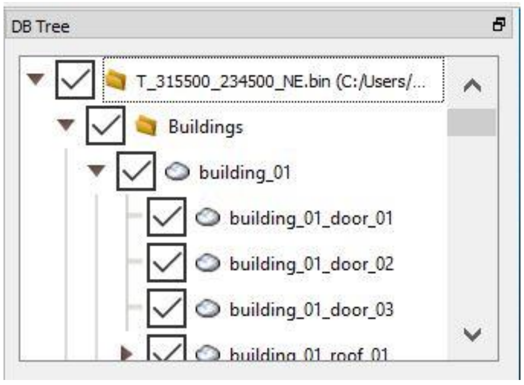


Figure 39 Metropolis model registered with DublinCity LiDAR data using CloudCompare


DB Tree

- T_315500_234500_NE.bin (C:/Users/...
  - Buildings
    - building_01
      - building_01_door_01
      - building_01_door_02
      - building_01_door_03
      - building_01_roof_01

Figure 40 Semantic labels for LiDAR data

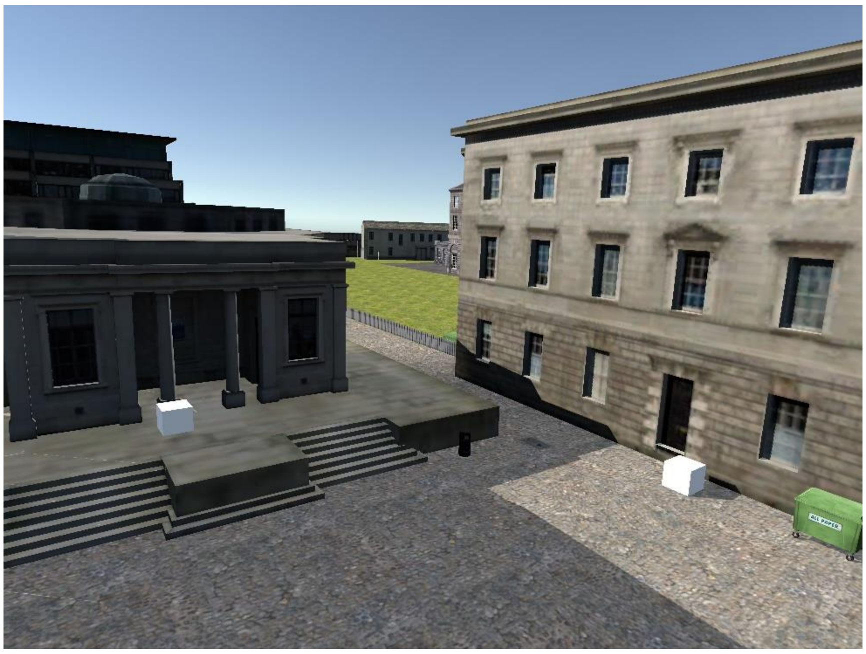
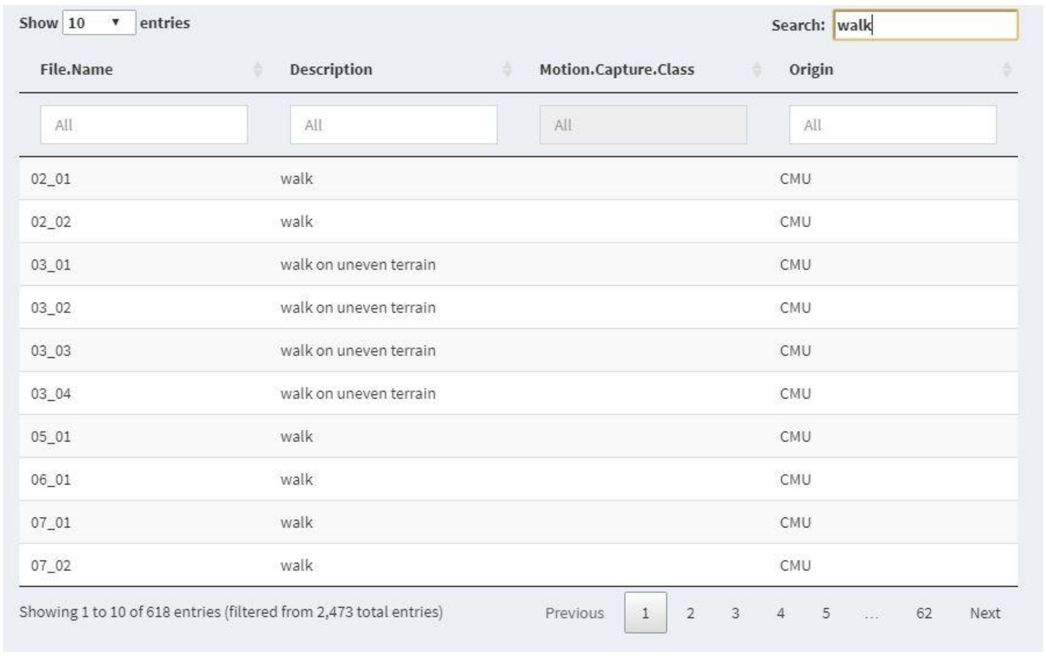


Figure 41 Door locations semantically annotated from labelled dataset (cube shows physical location) using the registered models in Figure 39

2. We used the animation database search tool in section 4.3.2 to identify and import suitable animations. These included animations that match the descriptions "walk", "idle" and "run", demonstrated in Figure 42 and Figure 43. Figure 43 shows the animations imported with the "humanoid" template in Unity Game Engine.


Show 10 entries

Search: walk

File.Name	Description	Motion.Capture.Class	Origin
All	All	All	All
02_01	walk		CMU
02_02	walk		CMU
03_01	walk on uneven terrain		CMU
03_02	walk on uneven terrain		CMU
03_03	walk on uneven terrain		CMU
03_04	walk on uneven terrain		CMU
05_01	walk		CMU
06_01	walk		CMU
07_01	walk		CMU
07_02	walk		CMU

Showing 1 to 10 of 618 entries (filtered from 2,473 total entries)

Previous 1 2 3 4 5 … 62 Next

Figure 42 Semantically annotated walk animations searched for.

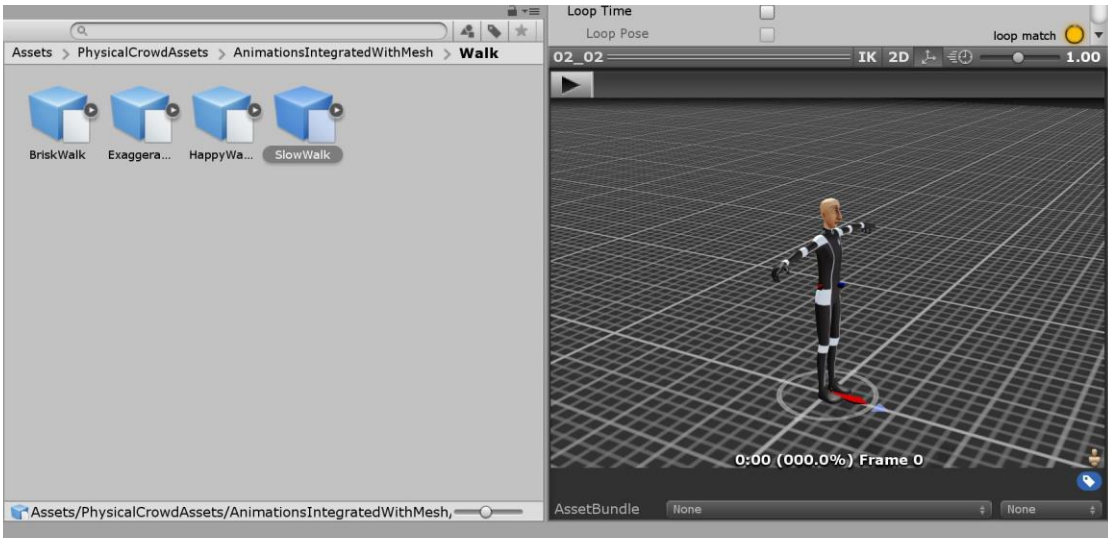

Assets > PhysicalCrowdAssets > AnimationsIntegratedWithMesh > Walk
BriskWalk
Exaggera...
HappyWa...
SlowWalk
Assets/PhysicalCrowdAssets/AnimationsIntegratedWithMesh/
Loop Time
Loop Pose
loop match
02_02
IK
2D
1.00
0:00 (000.0%) Frame 0
AssetBundle
None
None

Figure 43 Walk animations imported to Unity scene.

3. We placed crowd members at their starting points. They were partitioned into "static" and "mobile" groups. The static groups represent irresponsible citizens who are loitering and not following social distancing guidelines and the members in the mobile groups can move around and do follow the guidelines. Figure 44 shows the static groups in initially placed in the Metropolis front square scene.

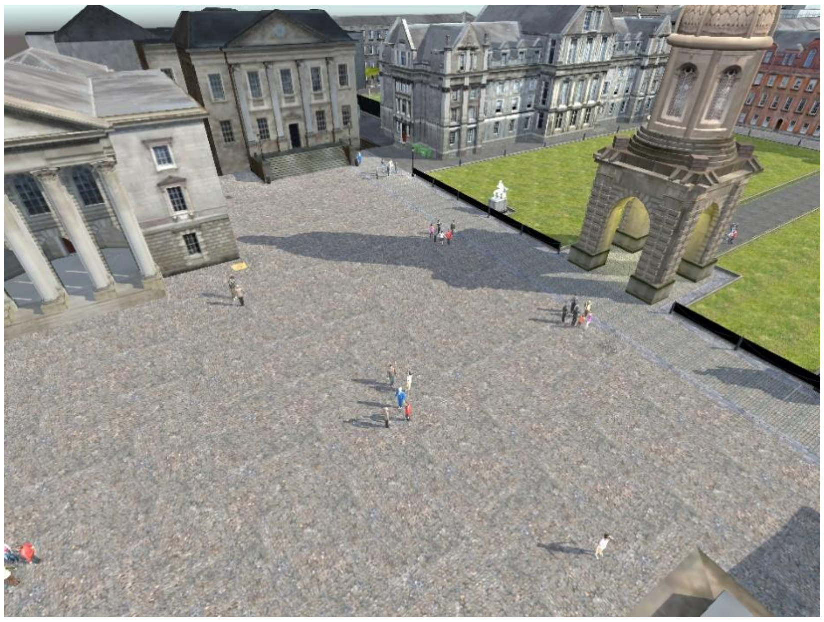


Figure 44 Static groups placed in scene

4. We created goal points and configured obstacles to be used with the potential field module. Static crowd members were treated as obstacles to ensure they would be avoided for social distancing purposes, shown in Figure 45. The purple circles in Figure 45 are visualisations of the repulsive forces attached to the static crowd members.

5. Paths were calculated and visualised for each agent using the potential field module, also shown in Figure 45. The repulsive forces, visualised with purple circles, push away the agents path. Three goal locations are specified, which first guides the crowd member to a water fountain, then through the Campanile arch and finally bring the crowd member mid-way up the central path. The path is visualised in yellow in Figure 45.

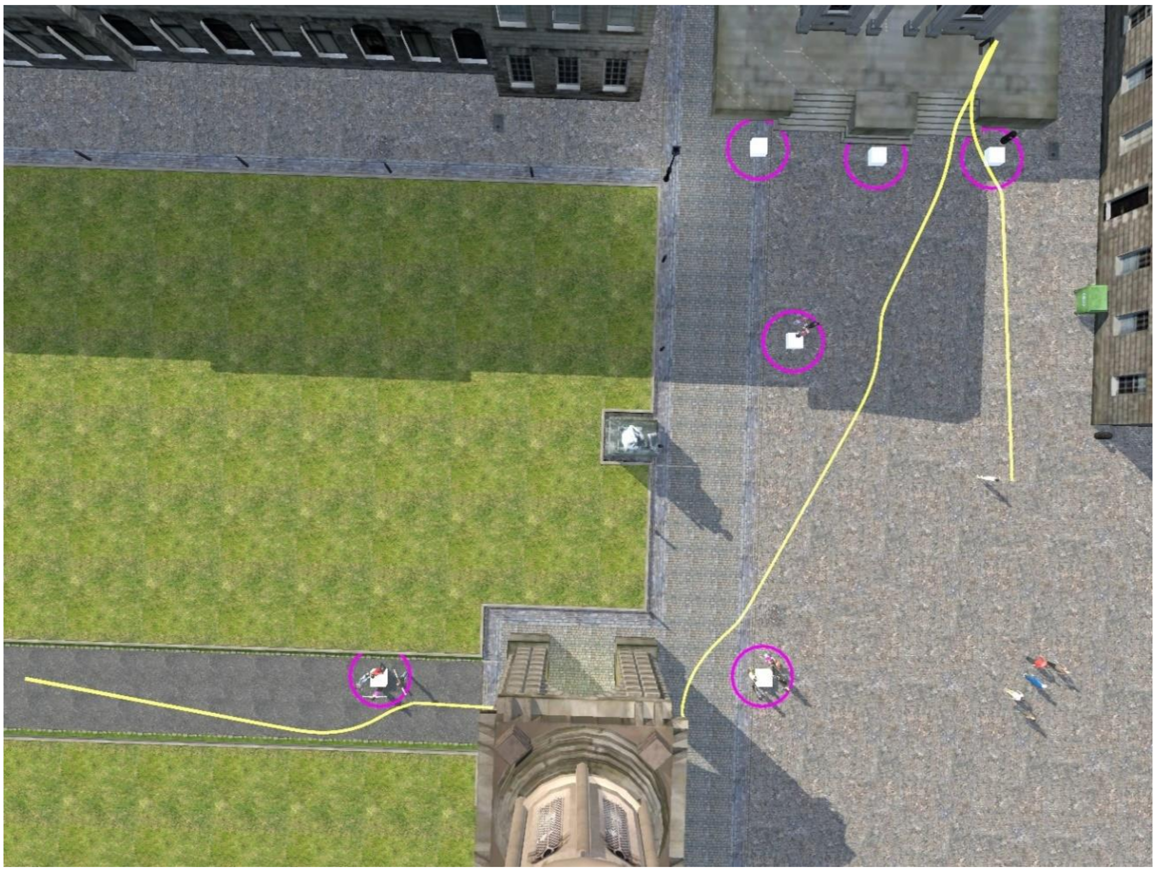
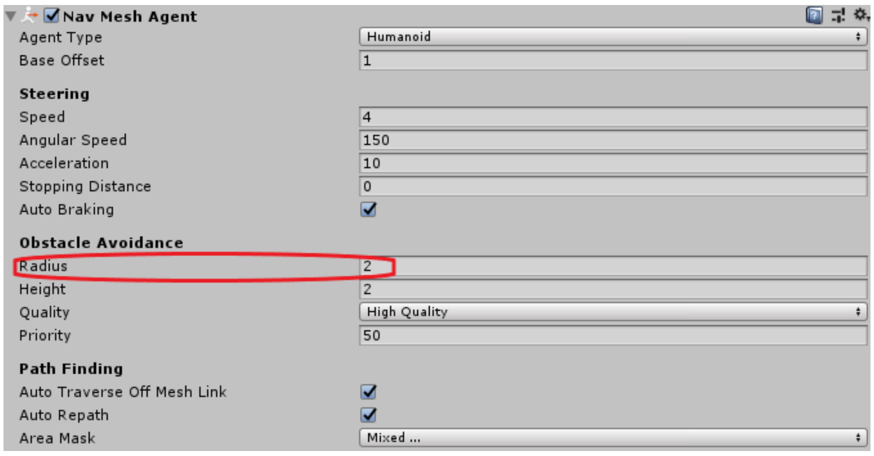


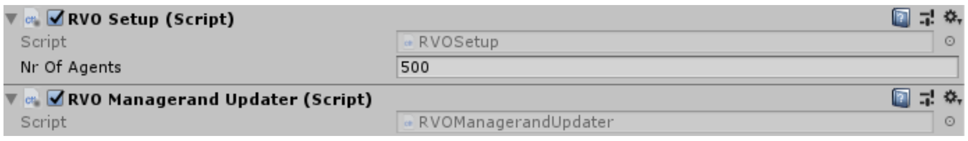
Figure 45 Potential field configured and visualised, with auto-generated sample path for one crowd member visualised

6. A component to perform local avoidance was added to the mobile crowd members and a global local avoidance manager was created. The local avoidance radius can be specified, as shown in Figure 46.



Nav Mesh Agent	
Agent Type	Humanoid
Base Offset	1
Steering	
Speed	4
Angular Speed	150
Acceleration	10
Stopping Distance	0
Auto Braking	✓
Obstacle Avoidance	
Radius	2
Height	2
Quality	High Quality
Priority	50
Path Finding	
Auto Traverse Off Mesh Link	✓
Auto Repath	✓
Area Mask	Mixed ...

Figure 46 Local avoidance module of crowd member with 2-unit radius as a pre-set for distancing



RVO Setup (Script)	
Script	RVOSetup
Nr Of Agents	500
RVO Managerand Updater (Script)	
Script	RVOManagerandUpdater

Figure 47 RVO2 global manager to keep track of and update crowd member position and velocity

7. Semantic animation triggers were created at suitable locations, including photograph points, water fountains and observation points. The associated animations were configured in the corresponding animation controllers. Figure 48 shows a crowd member interacting with a "drink" semantic animation trigger, which was placed in front of a water fountain. Once the animation has been triggered, the crowd member can continue on their path, illustrated in Figure 49.

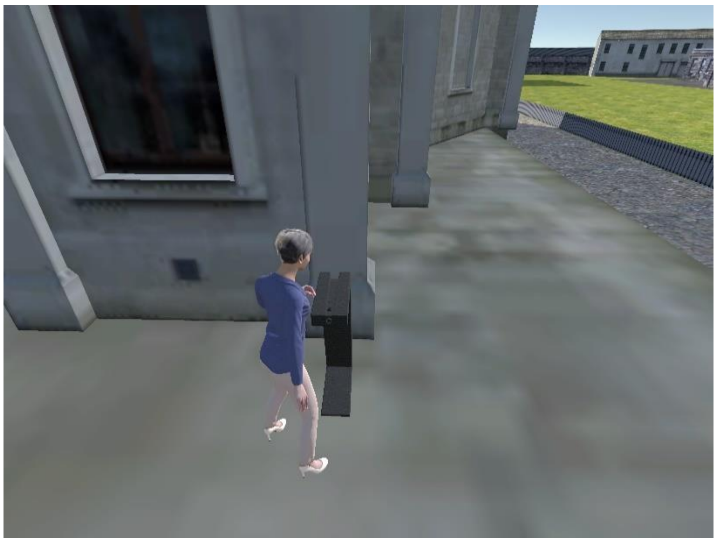


Figure 48 Crowd member triggers semantic "drink" animation

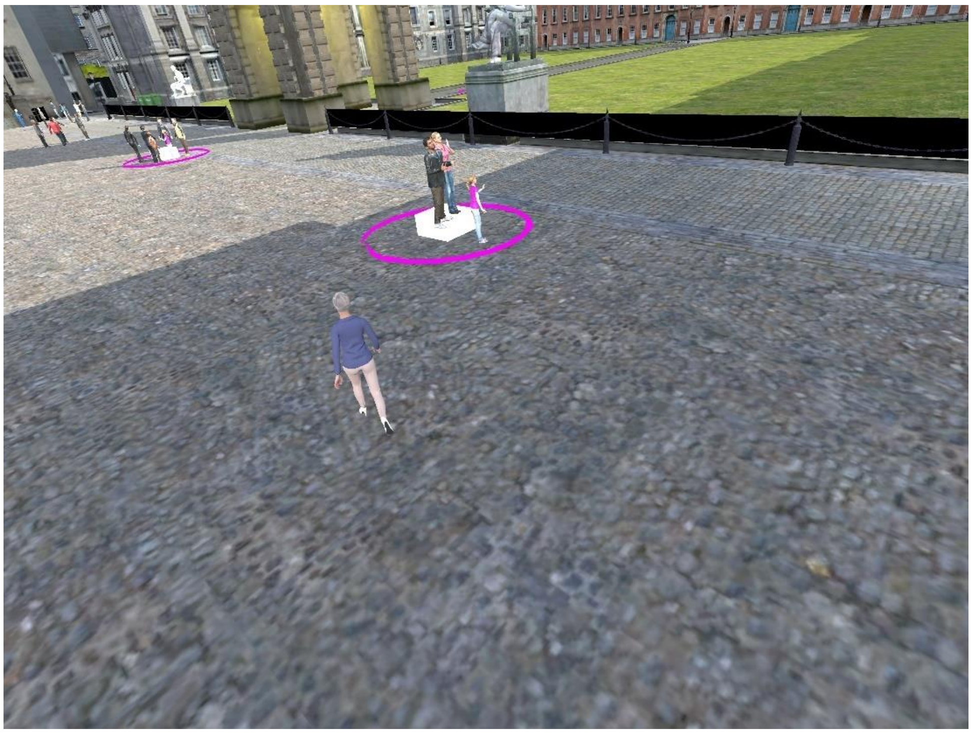


Figure 49 Once animation has played, the crowd agent continues

8. The real-time stylistic animation component in Section 4.4.4 was attached to crowd members whose animations needed to be altered.
9. We repeated parts of steps 1-9 as often as was necessary until the scene looked "believable".

Once each of these steps were carried out, we ran the scene. Figure 50 - Figure 52 show some results of the completed scene.

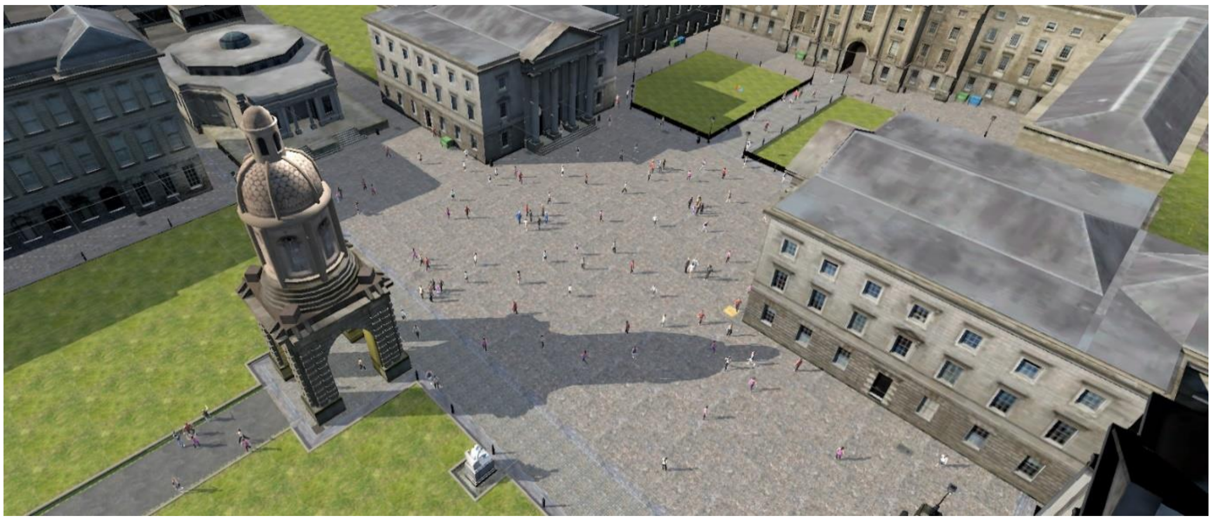


Figure 50 Completed social distancing scene top view

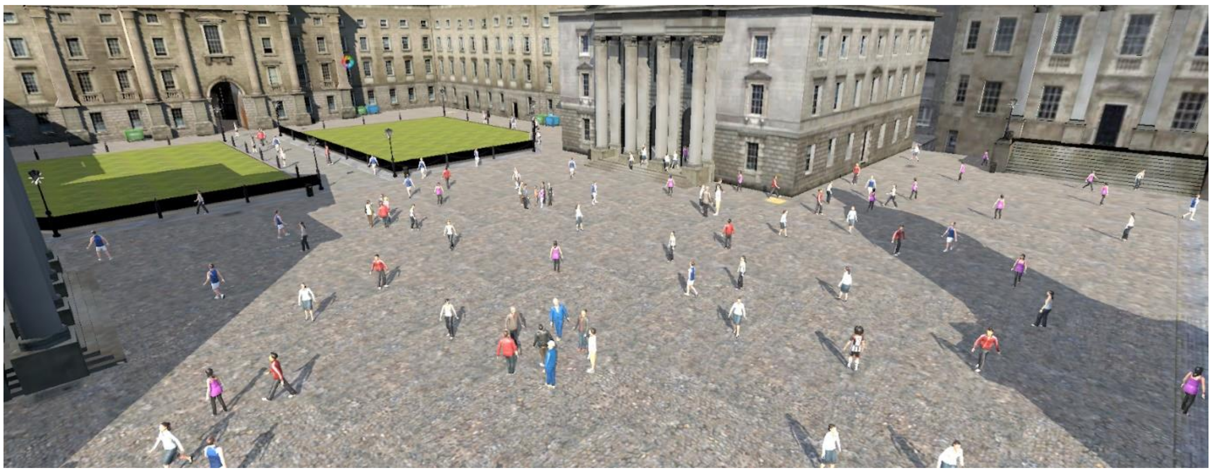


Figure 51 A still from the final social distancing crowd scene

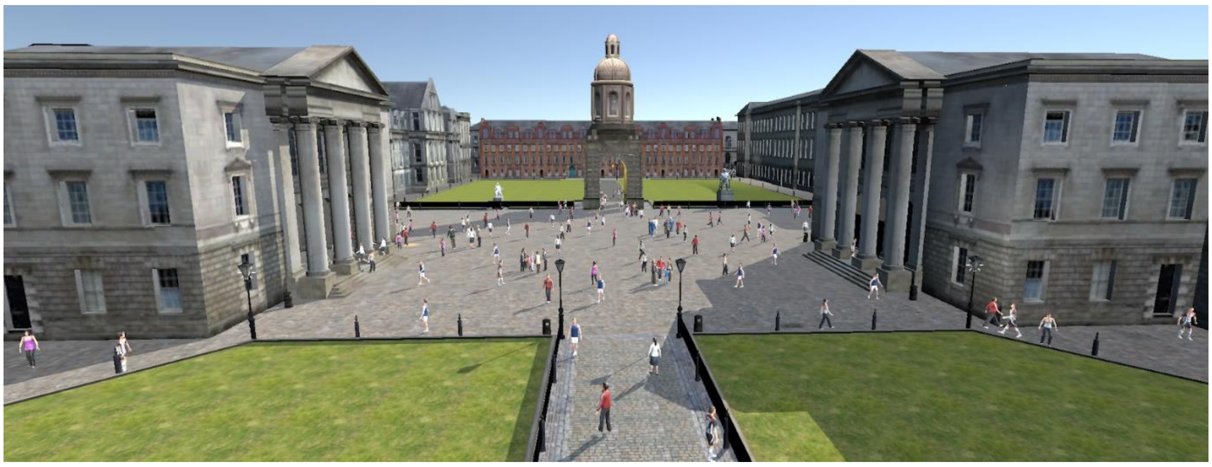


Figure 52 A still from the final social distancing crowd scene

4.5.2 Pedestrian Scene

Our second use case describes a generic pedestrian scene, which is frequently used as a development and evaluation platform for crowd simulation research [17], [31], [32]. We made use of the Love and Fifty Megatons (LAFM) [11] assets for this scene, developed and released for research purposes by

Filmakademie (FA). We deemed the assets suitable for our pedestrian use-case as they contain a long street with side-streets and obstacles such as benches and trees. This offers an ideal way to test high-level navigation capabilities. Figure 53 and Figure 54 show the environment model without any crowd members.

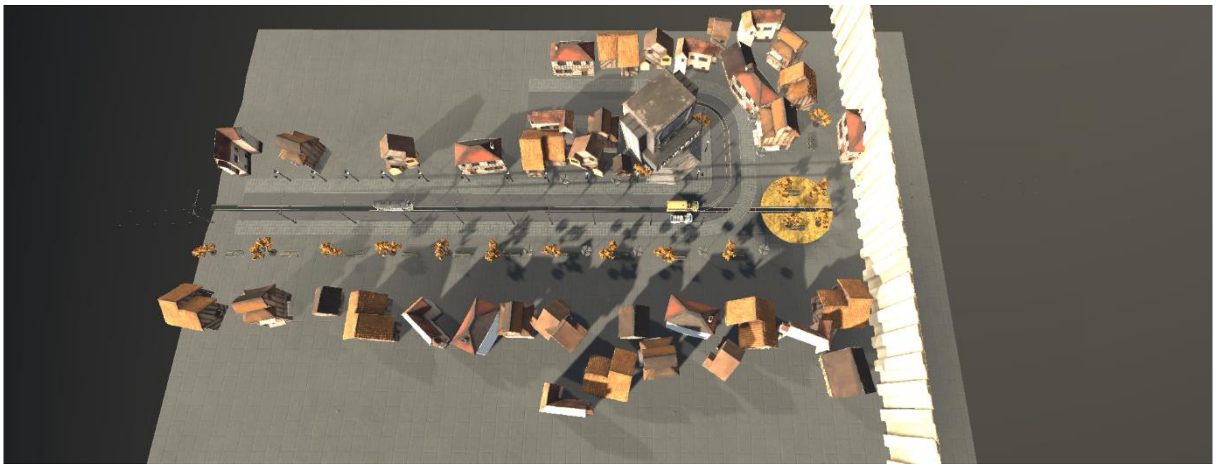


Figure 53 LAFM asset top view

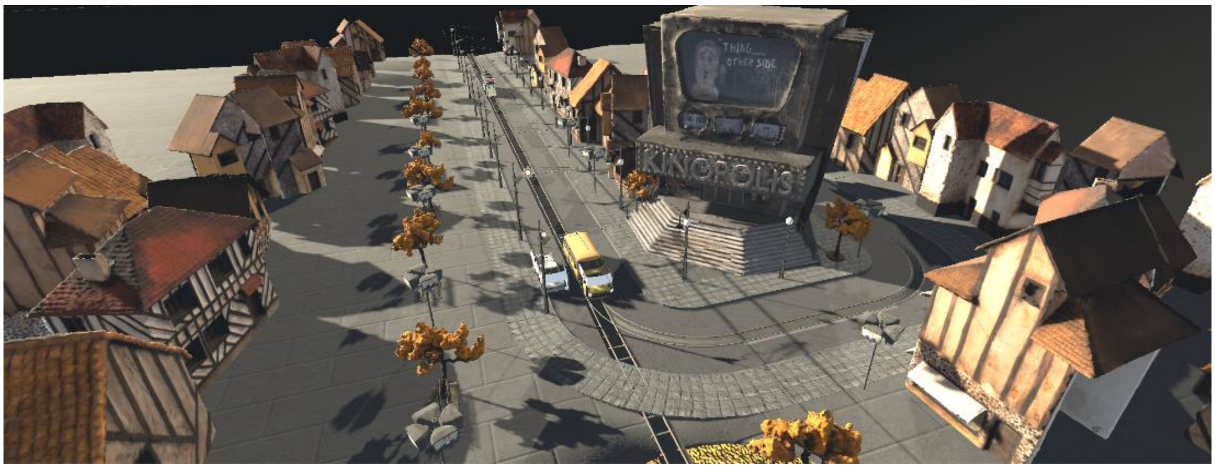


Figure 54 LAFM street view

We implemented this scene in a manner like the first, which the reader may refer to for the steps to reach the desired results. Figure 55 – Figure 57 show some stills of the end result for this scene.

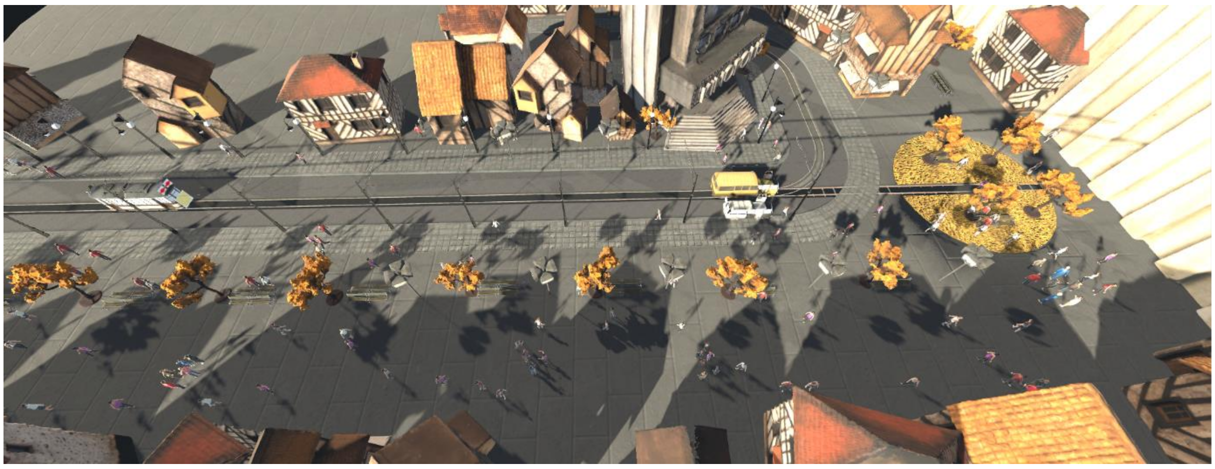


Figure 55 Pedestrian scene top view

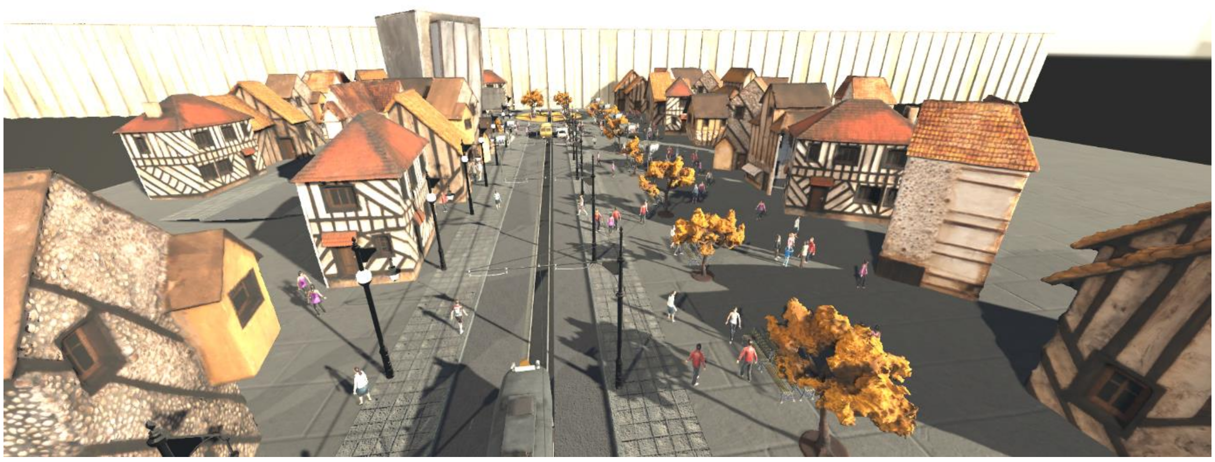


Figure 56 A still from the completed Pedestrian scene

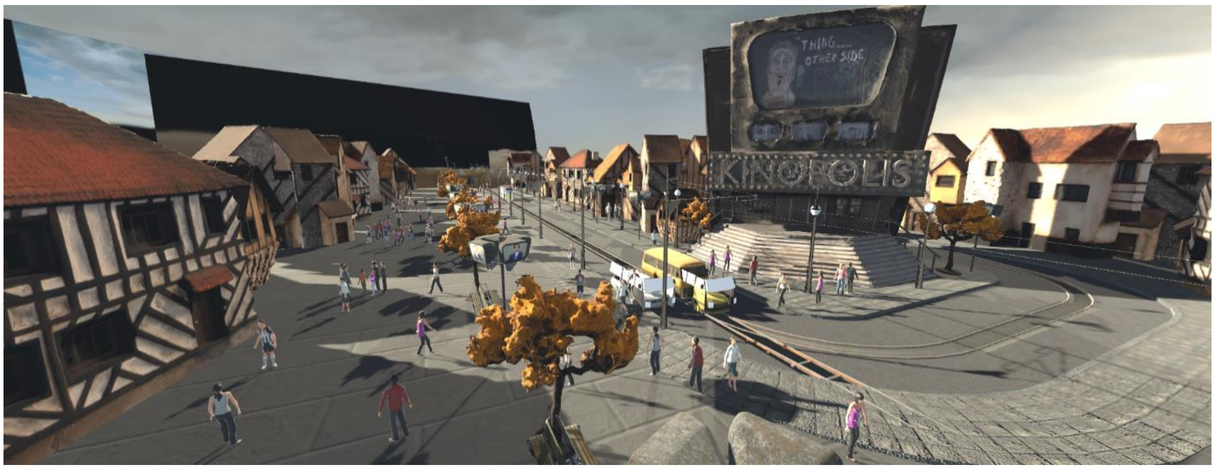


Figure 57 A still from the completed Pedestrian scene

4.5.3 Commuter Scene

Our final use-case is a commuter scene. As with the other use-cases, this is a general scene which made it easy to informally evaluate performance and identify inadequacies while developing the system. We chose this scene to complement the pedestrian scene, since they are semantically similar – in both scenes crowd members move along a pavement to a destination. This allowed us to test the re-targeting capabilities of our system. We chose to use assets developed as part of D6.3, which were semantically labelled using the DublinCity LiDAR data set [29], [30]. We took advantage of the pipeline described in D6.3, which allowed us to process the semantic assets to create a useable environment in Unity. Figure 58 and Figure 59 show the environment model without any crowd members. The scene is colour coded by the semantic asset label, where Orange = Building, Red = Ground, Grey = Sidewalk, Green = Vegetation.

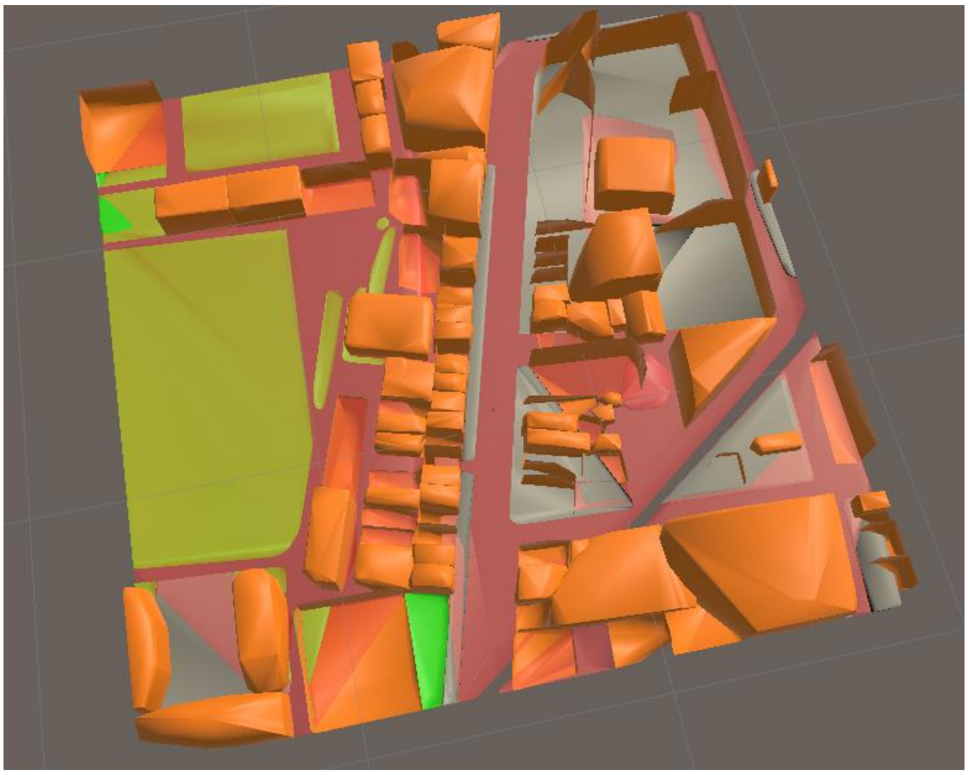


Figure 58 LiDAR scene, generated using the pipeline documented in D6.3

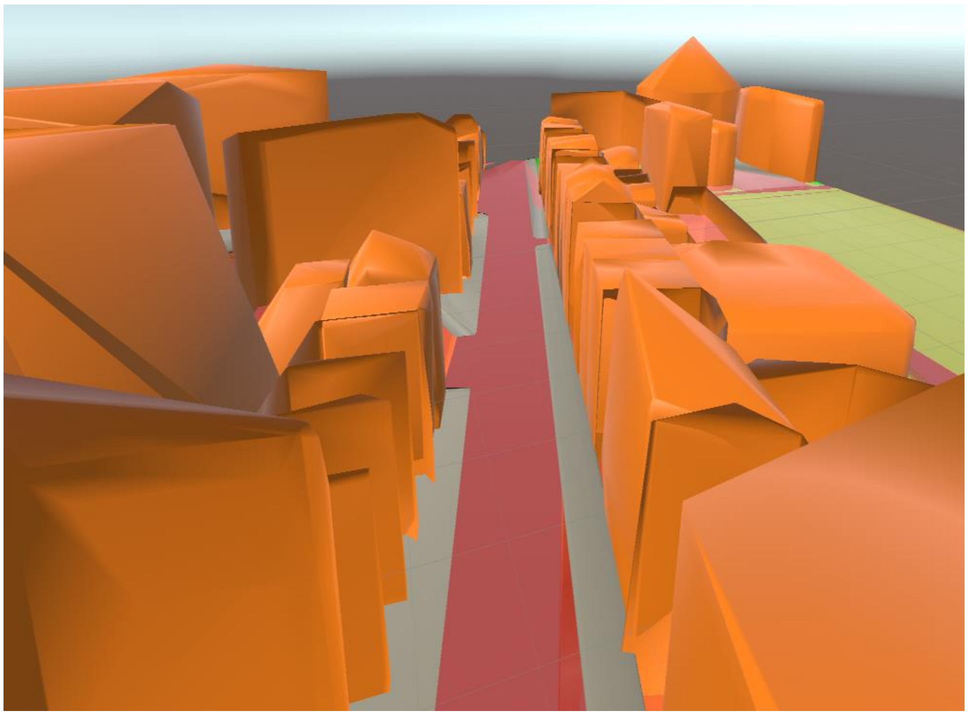


Figure 59 Street view of LiDAR scene.

The steps to implement this scene differ to that for the previous two, since we re-targeted the existing pedestrian crowd to this scene. We outline the re-targeting steps here:

1. This scene already has semantic labels, noted above. These were processed using the semantic data annotation module in 4.3.1 to provide JSON files according to a set schema. The colours in the scene correspond to: Building – Orange, Ground – Red, Sidewalk – Grey, Vegetation – Green.

2. Each of the pedestrians were exported from the pedestrian scene and then imported into this scene. For Unity-to-Unity retargeting, this can simply be done via drag-and-drop. We anticipate that an extension of our work could specify a general serialization method, based on the USD format [33].
3. We imported additional animations for this scene that could not be re-targeted from the pedestrian scene. We also created extra crowd members that could not be re-targeted for the pedestrian scene.
4. For each of the pedestrians, we updated the configuration of the potential field by specifying the positions of the start and goal locations, as well as any obstacle locations. We also updated the position of semantic animation triggers as well as their associated animations.
5. We removed any unnecessary components for the re-targeted crowd members.
6. We calculated and visualised paths for each agent using the potential field module. We repeated parts of steps 3-9 as often as was necessary until the scene looked subjectively "believable".

Figure 60– Figure 62 show some stills while running this scene.

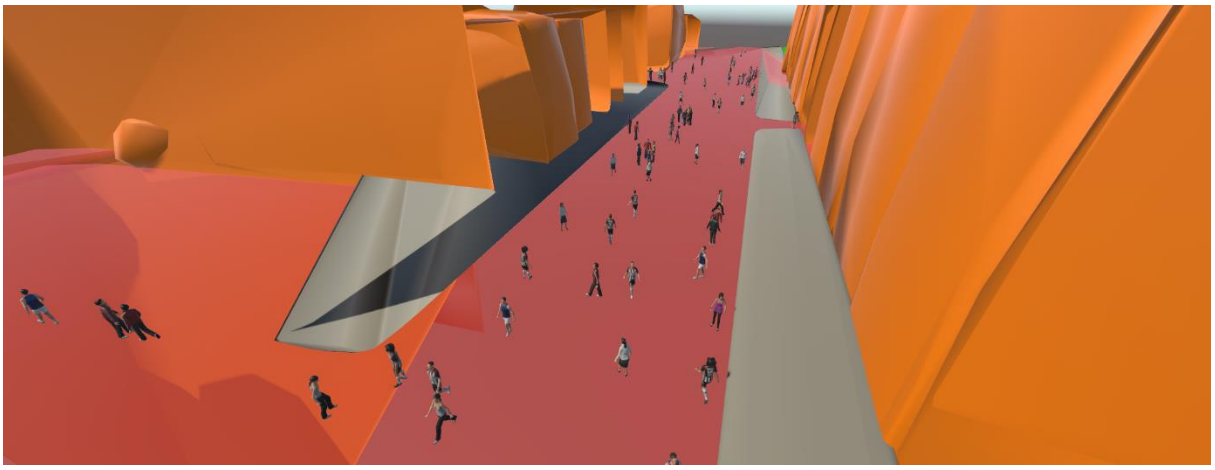


Figure 60 Completed re-targeted crowd simulation for LiDAR scene

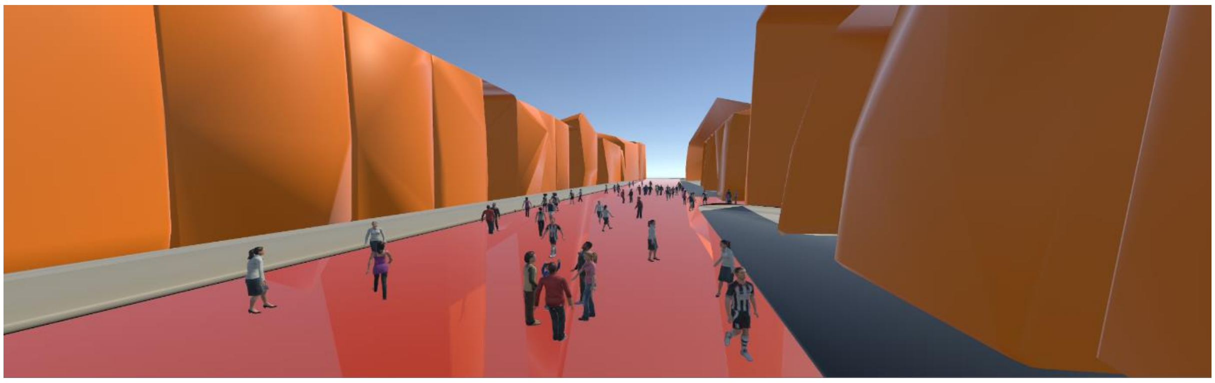


Figure 61 Completed re-targeted crowd simulation for LiDAR scene

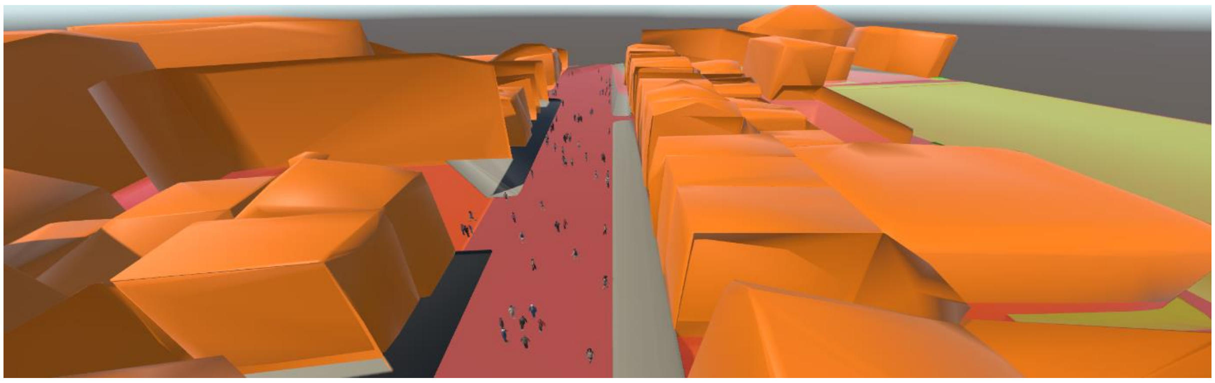


Figure 62 Completed re-targeted crowd simulation for LiDAR scene

4.6 Metrics and Evaluation

This section presents a set of metrics devised to give an end user an idea of how the system performs during run-time, as well as a plan for evaluating the framework. The evaluation is presented in two main parts:

1. Laboratory testing of quantitative metrics (system stress testing and FPS/memory).
2. A plan for an expert review of the of the tools documented in this deliverable by external professionals.

4.6.1 Evaluation Metrics and Methods for Laboratory Testing

The objective run-time performance of the modules developed in our system was recorded on a system in a research lab in Trinity College Dublin, where the frames per second and computation time by the system were recorded for a simple scene described in Section 4.6.1.2. The parameters varied as a part of this testing and their corresponding outputs are listed in Table 2 to Table 5. These tests were carried out to report the average performance that can be expected by the system in real-world usage. The specifications of the machine used to run the tests are reported in Table 1.

4.6.1.1 Software and Hardware Specifications

We recorded the metrics in the subsequent sections using the following specifications:

Hardware/ Software Component	Specification/ Version
CPU	Intel Core i7-6700K Processor 4.0 GHz
GPU	GeForce RTX 2080 Super (Base clock: 1650 MHz, 8 Gb of GDDR6 Memory and 3,072 CUDA cores)
Motherboard	Alienware 01NYPT
System Memory	DDR4 32 Gb
Operating System	Windows 10 Home
IDE	Visual Studio 2017 Version 15.9.17
Game Engine	Unity version 2018.4.12f1 (LTS)

Table 1 Hardware and Software specifications of PC used for evaluation

4.6.1.2 Stress Testing

We used Unity's in-built profiler to analyse the performance of number of crowd members in scene with regards to the modules' frames per second and computation time to estimate their individual performance. A basic crowd scene was created and the components we created were turned on and off. We then recorded the in-built Unity profiler output, which breaks down performance by component. We have considered only the major profiler attributes that cause impact to the performance for the stress testing (eg: Animation Triggers depend on both animation and physics; however, animation's overhead dominates as compared to physics). The measurements ignore the arbitrary quality of both the environment and the crowd member and only report on the performance of individual components.

	Number of Agents				
		100	500	1000	2000
FPS of approach	**Standard**	200	60	< 1	Scene not loading
	Object Pooling	250	100	61	29

Table 2 A comparison of time taken to load a basic scene using Object Pooling as opposed to the standard approach

	Number of crowd members in scene						
Stylistic Animations, Animation Triggers (Animation Profiler)		100	500	1000	5000	10000	20000
	Computation Time (ms)	0.49	1.55	3.21	17.32	33.75	66.34
	FPS	3500	920	630	60	32	14

Table 3 Stress testing results for Stylistic animations and Animation triggers

	Number of crowd members in scene						
Navigation Component (Script Profiler)		100	500	1000	5000	10000	20000
	Computation Time (ms)	3.17	9.63	11.75	69.38	145.65	273.70
	FPS	275	125	80	14	3	< 1

Table 4 Stress testing results for Navigation

	Number of crowd members in scene						
Local Avoidance (Physics Profiler)		100	500	1000	5000	10000	20000
	Computation Time (ms)	0.11	0.23	0.31	0.71	0.86	1.03
	FPS	9300	5800	3700	2500	1800	1200

Table 5 Stress testing results for Local Avoidance

Since our toolset provides the standalone potential field module to calculate the paths of crowd members, we also report on metrics associated with this module to give the end user an idea of expected performance. We developed our toolset so that computation is done before runtime, in anticipation of large crowd scenes that may require the majority of the system's runtime resources for rendering, collision detection, etc. This is not a necessity and the potential field path computation can be carried out at runtime instead if desired. This computation is performed asynchronously in our Unity implementation, which means that the user can continue developing the scene while paths are calculated in the background, minimising the overhead associated with this approach. For other systems that may use this module during run-time, we provide Table 6 as a guideline for expected performance. The two primary factors affecting performance are the distance to the goal and the number of repulsors in the field, which are varied. The repulsors were equally spaced between the start location and the goal location. Table 6 implies that a large amount of time is spent on initialisation of parameters, which could be a future optimisation.

	Distance to goal location					
Number of repulsors present in potential field		**10**	**15**	**30**	**80**	**200**
	1	0.93863	0.96933	0.931959	0.944465	0.930965
	2	1.057215	1.062002	1.029763	1.129277	1.417194
	3	1.05236	1.042283	1.062729	1.18819	1.475051
	4	1.15563	1.06063	1.072493	1.215221	1.620668
	5	1.260802	1.146646	1.158097	1.327577	1.750601

Table 6 Seconds taken to calculate a path using the potential field module as a function of distance to goal and number of repulsors present.

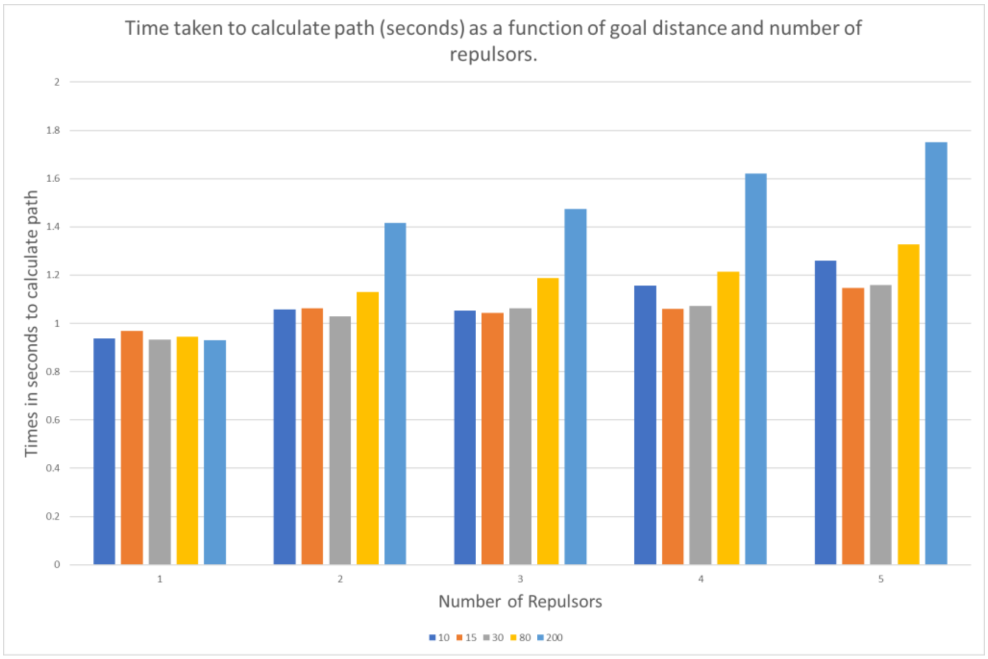


Figure 63 Graph of seconds taken to calculate a path using the potential field module as a function of distance to goal and number of repulsors present.

4.6.2 Description of Methodology for Expert Review

The aim of the expert study is to evaluate the time saving achieved by the crowd simulation tools developed by TCD for the SAUCE project, in comparison to Unity's built in tools. We hypothesise that our tool offers a time saving greater than 50% as compared to the default Unity tools. We will break down the time saving task-by-task mentioned in Section 4.6.4, which will allow us to identify the common areas of crowd simulation for which the developed toolset offers an advantage over the default tool set. This will enable us to provide an unbiased and structured evaluation of the crowd simulation work that we have done as part of the SAUCE project and ensure that we can meet requirements for the self-assessment plan set out in SAUCE D1.2 Self-Assessment plan [34]. The collected data will be statistically analysed and used to draw conclusions on our hypothesis for the potential time saving. A more detailed breakdown of the information that will be recorded is provided in section 4.6.5.

4.6.3 Modifications Due to Covid-19 Pandemic

Due to safety concerns in light of the Covid-19 pandemic, participants will not be invited in the V-SENSE lab to conduct the expert review, and as an alternative, arrangements have been made to conduct the study remotely. The whole procedure of the remote study is expected to take 10 weeks in total. 4 weeks out of those 10 weeks have been designated for recruitment of Subject Matter Experts (SME) [35] and the remote studies will be carried out in the remaining 6 weeks. The expert review will be conducted on a system in the V-SENSE lab of Trinity College Dublin, with the participants connecting to the system using a remote access software such as Teamviewer or Anydesk. Participants will be provided with the same system to minimise the number of external variables which can lead to inconsistent results. Participants will be provided with access to the system using a temporary password, and a team member will be online with them, available for remote support in case any assistance is required. We will consider Subject Matter Experts (SME) who will be recruited through word of mouth through existing contacts in the industry, relevant social media channels and direct contact through freelancing websites. The results of this expert review will be included in SAUCE deliverables D8.4 Report on Experimental Production Scenario Results and D8.5 Combined Evaluation Report.

4.6.4 Steps Involved in Expert Review

The study will follow a two-step strategy. The first involves creating a simulated crowd scene. The second involves retargeting the crowd to a semantically similar scene. We will ask the users to generate a crowd simulation using our toolset in a random 50% of the experiments (Approach A), and the remaining users will be instructed to use the default tools provided by Unity Game Engine (Approach B) to develop a similar crowd scene. Time measurements will be noted as described in section 4.6.2. The approaches will be marked with letters (A or B) to prevent any bias amongst the subjects.

Further, they will be provided with two documents:

1. A document which contains a high-level description/story for both scenes (Scene A & Scene B)
2. A document which provides the ordered steps the participants will have to complete. This serves as a suggestive guide on how to successfully develop the crowd simulation in Scene A and retarget it to Scene B.

The documents will be dependent on the approach assigned to the respective participant. Additionally, depending on that approach, participants will be given a brief video tutorial on how they can develop a crowd simulation for the study. We will then describe the experiment and show them the interface that they will be using, which is Unity Game Engine. The tasks in the steps document are a decomposition of the larger task of creating a simulated crowd. They will be allowed to ask questions about how they can complete these tasks. Specific information on what the result they should work towards will also be provided. We will then inform them that the experiment is starting, and they will be provided remote access to a machine in the V-SENSE office in TCD, which they will be using to complete the tasks. We will then start a timer and begin recording the screen, keystrokes and mouse clicks on the machine. The high-level tasks that the participants will need to complete are as follows:

1. Asset import and preparation
2. Creation of each crowd member type (random, goal-oriented, static)
3. Divide the agents into groups according to behaviour and attach associated navigation components as necessary
4. Configure non-static agents with navigation components
5. Configure animations for each of the agents
6. Run the simulation and modify parameters as necessary until the result is met

Once the user has completed the experiment, they will press a button which will stop the timer and terminate the screen recording session. They will then be asked to complete a subjective UMUX (The Usability Metric for User Experience Questionnaire) [36] to evaluate Usability and a UEQ (User Experience Questionnaire) [37] to evaluate User Experience of the provided system. The purpose of recording the UMUX and UEQ responses is to compare the qualitative performance of our toolset as compared to the default Unity tools. Qualitative performance measurements provides us with data related to subjective experience of the participants, and this feedback is useful for improving the user experience of the participants.

4.6.5 Measurements

For each participant, we will be measuring and recording the following information during this experiment:

1. Time on task for each of the steps 1-6 identified in section 4.6.4, calculated from the recording of the screen.
2. Keystrokes and mouse clicks throughout the course of the experiment.
3. The result achieved by each participant, in the form of a Unity Scene file.

A screening expert self-identification questionnaire will be presented to the users to determine the skill level of the participant. As discussed in section 4.6.3, only self-identified experts will be allowed to participate in the study. Once the user study begins, we will collect the keystrokes and mouse clicks so that we can record any periods of inactivity (e.g. if the user goes to the bathroom) so that they can be subtracted from the overall time taken. We will record this data in order to report on the time saving gained by using our tools. We will also be recording the data for each of the fields in the UMUX questionnaire and the UEQ questionnaire that each participant will be asked to fill in on the completion of the experiment. We will save the final scene file containing the developed crowd in order to ensure a level of consistency between participants. All this data will be created and saved internally on a TCD computer, via a remote access link. Each of the sessions will be anonymised with a randomly assigned ID.

5 Conclusion

This work has served as an investigation into how crowds can be developed to be reusable across scenes that are semantically similar. There are numerous avenues for exploration on this front and we note here that we only investigated a subset. Our approach to high-level behaviours took an approach referred to in the literature as *hybrid* [38]*,* but other approaches are promising in relation to semantic data use, for example *microscopic* approaches. We view the use of semantic data in crowd scenes to generally be a strong candidate for further research and we anticipate that this work will serve as a proof-of-concept which may spur further work in this area and for virtual productions in general.

We found that developing crowd simulations in general is currently taken as a specific endeavour and that they typically require the use of niche software (e.g. Massive, Menge, Goalem, Legion). Many of these are proprietary and cannot be integrated across platforms, which presents an immediate challenge in that boiler-plate code often needs to be written to achieve some relatively simple and routine tasks. This also creates a problem in that these approaches and technologies can be difficult to extend and customize, which creates a barrier for smaller studios with limited resources. To this end, we intend that this document and the associated techniques may help guide smaller studios in developing crowd simulation pipelines and that it may offer a clear strategy for developing crowds that can be repurposed, saving time and money. We plan that smaller studios would be the primary users of the technologies in this deliverable, although we hope that the scope could extend as a proof-of-concept for larger studios.

Our experience of actually creating development crowd scenes was a largely repetitive one, which led us to break down the process into individual modules, documented in Section 4. For each logical part of the process, we created a module which carries out a specific action. These steps are demonstrated in the use cases in Section 4.5. We advocate this modular approach, as it provides flexibility when creating crowd members. When some undesired behaviour is encountered, we can simply turn off the relevant component. We would encourage future work to treat crowd simulation as a set of components, so that some cross-platform modular pipelines may start to emerge. A good example of this is the RVO-2 local avoidance library [23], which is available in multiple languages and can be easily integrated with many different platforms.

We found that using semantic data to drive crowd behaviour can be an effective time saving tool, especially when the semantic data is readily available and in a suitable format. This implies that to some degree, this work depends on the automation of the semantic classification of assets, which is an active research topic in the SAUCE project, covered by Work Package 4. This leads to the conclusion that a crowd simulation using semantic data is possible, as documented in the deliverable, but it depends on the quality and detail of the semantic data. AI has seen a recent surge in research and now many mature and robust methods exist for tasks such as path-finding and general decision-making. Techniques such as reinforcement learning (RL) allow an autonomous agent to make decisions based on the perception of its surroundings, which we anticipate will become a more prominent method in crowd simulation research once semantically annotated assets replace regular assets.

A significant hurdle that we experienced in this work was devising a sufficient set of metrics to adequately evaluate the effectiveness of the tool set that we developed. Generally speaking, the overall quality of an arbitrary crowd simulation is something that cannot be measured objectively, since some aspects depend on subjective qualities. As noted in section 4.1, many publications don't attempt to define directly how to evaluate the overall quality of a crowd simulation can be measured, but rather break it down into components that can be more easily evaluated [12]. Since we are primarily interested in the time-saving offered by our toolset through re-targeting, we created an evaluation strategy that attempts to isolate this time saving. This will be recorded and statistically analysed in a user evaluation study, which will be reported later in the SAUCE project (D8.4 *Report on Experimental Production Scenario Results* and D8.5 *Combined Evaluation Report*). The planning of the evaluation of this work was a non-trivial task and we intend that the methods and metrics devised can be built upon in future work to move towards a standard conventional set of test scenarios and metrics for researchers involved in crowd simulation.

6 References

[1] N. &. A. J. &. K. M. &. B. N. Pelechano, Simulating Heterogeneous Crowds with Interactive Behavior, A K Peters/CRC Press, 2016.

[2] A. Bielik, "'I Am Legend': Apocalypse Now in Manhattan," 14 12 2007. [Online]. Available: https://www.awn.com/vfxworld/i-am-legend-apocalypse-now-manhattan. [Accessed 5 6 2020].

[3] D. Greer, "https://www.rocketstock.com/blog/crowd-control-the-vfx-behind-dynamic-crowd-simulations/," 16 10 2015. [Online]. Available: https://www.rocketstock.com/blog/crowd-control-the-vfx-behind-dynamic-crowd-simulations/. [Accessed 4 6 2020].

[4] "Massive Crowds in World War Z," 2017. [Online]. Available: http://www.massivesoftware.com/wwz.html. [Accessed 4 6 2020].

[5] [Online]. Available: http://www.massivesoftware.com/. [Accessed 4 6 2020].

[6] "Menge Home," 2013. [Online]. Available: http://gamma.cs.unc.edu/Menge/. [Accessed 2 6 2020].

[7] "Welcome to uCrowds," [Online]. Available: https://ucrowds.com/. [Accessed 5 6 2020].

[8] N. . Kraayenbrink, J. . Kessing, T. . Tutenel, G. d. Haan, F. . Marson, S. R. Musse and R. . Bidarra, "Semantic crowds: reusable population for virtual worlds," *Procedia Computer Science,* vol. 15, no. , pp. 122-139, 2012.

[9] European Commission, "Smart Asset re-Use in Creative Environments | SAUCE Project | H2020 | CORDIS | European Commission," 2020. [Online]. Available: https://cordis.europa.eu/project/id/780470. [Accessed 14 June 2020].

[10] C. O. a. C. Ennis, "Metropolis: multisensory simulation of a populated city," in *International Conference on Games and Virtual Worlds for Serious Applications*, 2011.

[11] Filmakademie, "Love And Fifty Megatons," [Online]. Available: https://www.loveandfiftymegatons.com/. [Accessed 1 6 2020].

[12] N. A. J. M. a. B. N. I. Pelechano, "Virtual Crowds: Methods, Simulation, and Control," *Animation, Synthesis Lectures on Computer Graphics and Animation,* pp. 1-176, 2008.

[13] N. I. B. R. B. J. C. A. J. M. S. J. &. P. M. S. Badler, "Real Time Virtual Humans," in *Proceedings ofThe Fifth Pacific Conference on Computer Graphics and Applications*, 1997.

[14] U. Technologies, "Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations," 2020. [Online]. Available: https://unity.com/. [Accessed 14 June 2020].

[15] R. E. da Silva, J. Ondrej and A. Smolic, “Using LSTM for Automatic Classification of Human Motion Capture Data,” in *14th International Conference on Computer Graphics Theory and Applications 2019*, 2019.

[16] K. M. L. S. H. G. A. K. W. B. L. E. K. a. S. T. Howie Choset, Principles of Robot Motion, MIT Press, 2005.

[17] S. C. Z. P. Adrien Treuille, “Continuum crowds,” *ACM Transactions on Graphics,* vol. 25, no. 3, 2006.

[18] A. Zeileis, “strucchange: Testing, Monitoring, and Dating Structural Changes,” [Online]. Available: https://cran.r-project.org/web/packages/strucchange/index.html. [Accessed 6 6 2020].

[19] J. a. P. P. Bai, “Computation and analysis of multiple structural change models,” *Journal of Applied Econometrics,* vol. 18, no. 1, pp. 1-22, 2002.

[20] Unity Technologies, “Unity - Manual: Inner Workings of the Navigation System,” 2020. [Online]. Available: https://docs.unity3d.com/Manual/nav-InnerWorkings.html. [Accessed 14 June 2020].

[21] M. C. L. a. D. M. Jur van den Berg, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *IEEE Int. Conf. Robotics and Automation*, Pasadena, 2008.

[22] J. v. d. B. S. J. G. S. C. S. P. M. C. L. a. D. M. Jamie Snape, “Independent navigation of multiple robots and virtual agents,” in *9th Int. Conf. Autonomous Agents and Multiagent Systems*, Ontario, 2010.

[23] J. Snape, “RVO2 Library - What is New in RVO2 Library,” 2012. [Online]. Available: http://gamma.cs.unc.edu/RVO2/documentation/2.0/whatsnew.html. [Accessed 14 June 2020].

[24] Unity Technologies, “Unity - Manual: Animation Controller,” 2020. [Online]. Available: https://docs.unity3d.com/Manual/class-AnimatorController.html. [Accessed 14 June 2020].

[25] Unity Technologies, “Unity - Manual: Working with humanoid animations,” 2020. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/AvatarCreationandSetup.html. [Accessed 14 June 2020].

[26] Unity Technologies, “Unity - Manual:,” 2020. [Online]. Available: https://docs.unity3d.com/462/Documentation/Manual/MecanimAnimationSystem.html. [Accessed 14 June 2020].

[27] Unity Technologies, “Unity - Manual: Understanding Automatic Memory Management,” 2020. [Online]. Available: https://docs.unity3d.com/Manual/UnderstandingAutomaticMemoryManagement.html. [Accessed 14 June 2020].

[28] S. J. G. B. Z. a. D. M. S. Curtis, “Virtual Tawaf: A case study in simulating the behavior of dense, heterogeneous crowds,” in *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, 2011.

[29] I. &. R. S. &. R. A. &. C. A. &. S. A. &. D. S. R. &. R. M. Zolanvari, “DublinCity: Annotated LiDAR Point Cloud and its Applications.,” in *British Machine Vision Conference*, (2019).

[30] D. &. A. S. &. V. A.-V. &. T.-H. L. &. G. H. Laefer, “2015 Aerial Laser and Photogrammetry Survey of Dublin City Collection Record,” 2017.

[31] M. a. K. L. a. G. M. Sung, “Fast and Accurate Goal-Directed Motion Synthesis for Crowds,” in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Los Angeles, 2005.

[32] D. L. B. A. J. a. T. W. Helbing, “Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions.,” in *Transportation Science*, 2005.

[33] Disney Pixar, “Introduction to USD,” 2017. [Online]. Available: https://graphics.pixar.com/usd/docs/index.html. [Accessed 15 June 2020].

[34] SAUCE, “D1.2 Self-Assessment Plan,” [Online]. Available: https://www.sauceproject.eu/dyn/1560417885361/D1.2-Extract.pdf.

[35] K. K. William W. Cotterman, “User Cube: A Taxonomy of End Users.,” *Communications of the ACM (COMMUN ACM) ,* 1989.

[36] K. Finstad, “The Usability Metric for User Experience,” *Interacting with Computers (INTERACT COMPUT) ,* 2010.

[37] M. Schrepp, “User Experience Questionnaire,” [Online]. Available: https://www.ueq-online.org/.

[38] K. S. S. a. S. H. Ijaz, “A Survey of Latest Approaches for Crowd Simulation and Modeling Using Hybrid Techniques,” in *Proceedings of the 2015 17th UKSIM-AMSS International Conference on Modelling and Simulation*, 2015.

[39] Unity Technologies, “Unity - Manual: NavMesh Agent,” 2020. [Online]. Available: https://docs.unity3d.com/Manual/class-NavMeshAgent.html. [Accessed 14 June 2020].

[40] U. Technologies, “Unity - Manual: Building a NavMesh,” 2020. [Online]. Available: https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html. [Accessed 14 June 2020].